UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ademir Goulart

AVALIAÇÃO DE MECANISMOS DE COMUNICAÇÃO EM GRUPO PARA AMBIENTE WAN

Dissertação submetida à Universidade federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Luis Fernando Friedrich, Dr

Florianópolis, junho de 2002

AVALIAÇÃO DE MECANISMOS DE COMUNICAÇÃO EM GRUPO PARA AMBIENTE WAN

Ademir Goulart

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____
Prof. Luis Fernando Friedrich, Dr

_____
Prof. Fernando Álvaro Ostuni Gauthier, Dr

Banca Examinadora

_____
Prof. Luis Fernando Friedrich, Dr.

_____
Profa. Anita Maria da Rocha Fernandes, Dra

_____
Prof. Rômulo Silva de Oliveira, Dr.

_____
Prof. Vitório Bruno Mazzola, Dr.

Aos meus filhos Gabriel e Miguel

... aprendi que o mais valioso **não é o que temos** em nossas vidas,
mas sim **quem temos** em nossas vidas

AGRADECIMENTOS

A Universidade do Vale do Itajaí, por proporcionar o apoio necessário e o acesso aos laboratórios para a construção deste trabalho.

Ao meu orientador, Luis Fernando Friedrich, Dr., pelo seu empenho a fim de enriquecer o trabalho e sua pronta disposição para o atendimento deste orientando.

Ao Alex Kuhnen pelo auxilio na codificação dos programas de demonstração no ambiente WINDOWS em Visual C, especialmente no tratamento da interface gráfica.

Ao Daniel Amaral Manfredine da UNIVALI, Oficina do Design, Campus de Balneário Camboriu, pela sua ajuda no desenvolvimento e edição das figuras.

Ao amigo Neto, José Morelli Neto, pela ajuda prestada no Campus de Balneário Camboriu, seja configurando máquinas nos laboratórios para a execução dos testes ou emprestando o seu "LINUX particular" para hospedar um *daemon* SPREAD.

Aos colegas professores do Curso de Ciências da Computação da UNIVALI pelo incentivo.

A todos os demais, não nomeados, que de forma direta ou indireta contribuíram para a realização deste trabalho.

RESUMO

Este trabalho é um estudo sobre comunicação em grupo usada em sistemas distribuídos. É feita uma avaliação de diversos mecanismos de Comunicação em grupo existentes atualmente. Para uma análise mais detalhada, são selecionados três ambientes que trabalham tanto em redes locais como em redes de longa distância. Finalmente um dos mecanismos de comunicação em grupo é escolhido, o SPREAD, e submetido a testes de volume, funcionalidades e performance em um ambiente real de rede WAN. Também uma aplicação de gerenciamento de recursos em ambiente de sistemas distribuídos foi desenvolvida para ilustrar o uso deste mecanismo de comunicação em grupo. Concluí-se que o SPREAD tem boa performance e escalabilidade em ambiente WAN podendo ser usado plenamente em uma ferramenta de gerência de recursos em sistemas distribuídos.

Palavras chaves: Comunicação em grupo, gerência de recursos, sistemas distribuídos

ABSTRACT

This paper is a study about group communication used in distributed system. An evaluation of several mechanisms is made about group communication available at the moment. Three environments that work as in local area as wide area network are selected for a more detailed analysis. Finally, one mechanism of group communication is chosen, the SPREAD software, and it is evaluated concerning scalability, performance and functionality using a real environment of wide area network. An application of resource management for distributed systems was also developed to demonstrate the facilities of this group communication software. It was conclude that SPREAD have a good performance and scalability in WAN and can be used for developing resource management software in distributed systems.

Key words: Group Communication, resource management, distributed systems

SUMÁRIO

Lista de Figuras XI

Lista de Quadros XIII

Lista de Abreviaturas e Siglas XIV

1 INTRODUÇÃO 17

1.1 Objetivos 18

1.2 Motivação 19

1.3 Materiais e Métodos 19

1.4 Conteúdo do Documento 20

2 CONCEITUAÇÃO BÁSICA 21

2.1 Características dos Sistemas Distribuídos e Paralelos 21

2.2 Estratégias para Sistemas Distribuídos 22

2.2 Estratégias para Sistemas Distribuídos 23

2.2.1 Considerações Básicas 24

2.2.2 Distribuição para Informação e Compartilhamento de Recursos 25

2.2.3 Distribuição para Maior Disponibilidade e Performance 25

2.2.4 Distribuição para Modularidade 26

2.2.5 Distribuição para a Descentralização 26

2.2.6 Distribuição para Segurança. 26

2.3 Classificação de Sistemas Distribuídos e Paralelos 27

2.3.1 Multiprocessadores 29

2.3.2 Multicomputador 31

2.3.3 Cluster 33

2.4 Software em Sistemas Distribuídos 35

2.4.1 Sistemas Operacionais 35

2.4.2 Ambientes de Programação Paralela e Distribuída 39

2.4.3 DCE Ambiente de Computação Distribuída 40

2.4.4 Programação Distribuída Orientada a Objeto 42
2.4.4.1 CORBA - Common Object Request Broker Architecture 43
2.4.4.2 JAVA Remote Method Invocation 46
2.4.4.3 DCOM - Distributed Component Object Model 48
2.4.4.3.1 Monikers 50
2.4.4.3.2 Chamadas a Métodos Remotos 50
2.4.4.3.3 Coleta de Lixo 51
2.4.4.3.4 Modelo de *Thread* Suportado em DCOM 51
2.4.4.3.5 Segurança em DCOM 52
2.4.4.4 Pontos Chaves de CORBA, JAVA e DCOM 52
2.5 Comunicação nos Sistemas Distribuídos 54
2.5.1 Arquitetura do Protocolo TCP/IP 54
2.5.2 Modelo Cliente Servidor 57
2.5.3 RPC Chamada Remota de Procedimentos 59

3 COMUNICAÇÃO EM GRUPO 65
3.1 Introdução 65
3.2 Organização dos Grupos 67
3.3 Gerenciamento na Comunicação em Grupos 68
3.4 Propriedades da Comunicação em Grupos 71

4 MECANISMOS DE COMUNICAÇÃO EM GRUPO 76
4.1 Mecanismos de Partição Primária 76
4.2 Mecanismos Particionáveis 77
4.3 Comparativos 80

5 INTERGROUP 82
5.1 Características 82
5.2 Implementação 84
5.3 Resultado Prático da Instalação 85

6 JGROUP 88
6.1 Características 88
6.2 Implementação 92
6.3 Resultado Prático da Instalação 96

7 O MECANISMO DE COMUNICAÇÃO EM GRUPO SPREAD 100
7.1 Introdução 100
7.2 Arquitetura do Sistema SPREAD 102
7.3 Protocolos 106
7.3.1 Visão Geral 106
7.3.2 Disseminação de Pacotes e Confiabilidade 107
7.3.2.1 Protocolo *HOP* 109
7.3.2.2 Protocolo *RING* 112
7.3.3 Entrega de Mensagens, Ordenação e Estabilidade 115
7.4 Interface de Programação de Aplicação (API) 117
7.4.1 Tratamento de Buffer Insuficiente 118
7.4.2 Tipos de dados da API 120
7.4.3 SP_Funções 121
7.4.3.1 SP_connect 121
7.4.3.2 SP_disconnect 123
7.4.3.3 SP_join 123
7.4.3.4 SP_leave 124
7.4.3.5 SP_multicast e familia 125
7.4.3.6 SP_receive e SP_scat_receive 126
7.4.3.7 SP_equal_group_ids 132
7.4.4 Considerações sobre a API 132

8 AVALIAÇÃO DE DESEMPENHO DO MECANISMO SPREAD 134
8.1 Exemplo de Aplicação 134
8.1.1 Programa Cliente 135
8.1.2 Programa de Consulta 137
8.1.3 Programa gerador de Log de Conexão 142
8.2 Experimentação Prática 144
8.2.1 Configuração da Rede 146
8.2.2 Resultados das Medições 150
8.2.3 Gráficos Comparativos 156

9 CONCLUSÃO 160
9.1 Quanto ao uso em Aplicações 160

9.2 Quanto a Performance e Escalabilidade 160
9.3 Futuros Trabalhos e Melhorias 161

10 REFERÊNCIAS BIBLIOGRÁFICAS 163

ANEXO A Descrição do Conteúdo do CD que Acompanha este Trabalho 167

Lista de Figuras

Figura 1 Uma Taxonomia para Sistemas Distribuídos e Sistemas Paralelos 29
Figura 2 Sistema Distribuído Baseado em Barramento.....30
Figura 3 Comutação Crossbar 30
Figura 4 Rede Omega 31
Figura 5 Sistema Constituído por Computadores Interligados em Rede 31
Figura 6 Topologia em Grade 32
Figura 7 Topologia em Hipercubo.....32
Figura 8 Arquitetura Típica de um Cluster 33
Figura 9 Estrutura em Camadas do NFS 37
Figura 10 Um Multiprocessador com uma Única Fila de Processos Prontos 38
Figura 11 Modelo de Referencia OMG OMA.....43
Figura 12 Arquitetura JAVA RMI.....46
Figura 13 Ativação de Objetos Remotos em JAVA RMI 48
Figura 14 Conceito TCP/IP 56
Figura 15 O modelo Cliente Servidor.....57
Figura 16 Ambiente Genérico para Cliente Servidor.....58
Figura 17 Arquitetura Genérica Cliente Servidor.....59
Figura 18 Mecanismo de Chamada Remota a Procedimento.....60
Figura 19 Servidor de Requisição de Objetos 64
Figura 20 (a) Comunicação Ponto-a-Ponto (b) Comunicação Um-Para-Muitos.....65
Figura 21 (a) Grupo Fechado (b) Grupo Aberto.....67
Figura 22 (a) Comunicação em Grupo de Semelhantes 68
Figura 23 (a) *Multicasting* (b) *Broadcasting* (c) *Unicasting* 70
Figura 24 Diretório de Mensagens dos Usuários U3 e U4.....72
Figura 25 Ordenamento Causal.....74
Figura 26 Quatro Processos A,B,C e D e Quatro Mensagens. 75
Figura 27 Exemplo do Aplicativo de Troca de Mensagens Usando INTERGROUP 86
Figura 28 Arquitetura do JGROUP 91
Figura 29 Arquitetura do JGROUP em Máquinas JVM.....92
Figura 30 A Arquitetura do *Daemon* JGROUP.....94
Figura 31 Dois Possíveis Empilhamentos das Camadas de Gerenciamento de Grupos .96

Figura 32 JGROUP Exemplo Hello Server 97
Figura 33 Aplicação de Medida de Performance Servidor GMI 98
Figura 34 A Configuração de uma Rede de Longa Distância com SPREAD 103
Figura 35 A Arquitetura do SPREAD 104
Figura 36 A Interface de Programação de Aplicação API - SPREAD.....106
Figura 37 Um Cenário de Demonstração do Protocolo *HOP*.....109
Figura 38 O Protocolo *HOP* 112
Figura 39 O Protocolo *RING* 115
Figura 40 Programa de Consulta e Programas Clientes 135
Figura 41 Mensagem do Tipo Pop-Up Enviada a Todos os Clientes.....136
Figura 42 Tela de Comandos do Programa de Consulta 137
Figura 43 Tela para Envio de Mensagem Genérica.....138
Figura 44 Consulta de Micro Livre 139
Figura 45 Consulta se Determinado Software Está em Uso.....140
Figura 46 Consulta Tipo de Arquivo em Todos os Micros 141
Figura 47 Consulta Usuário 142
Figura 48 Relatório de Uso de um Micro.....143
Figura 49 Interligação dos Programas FLOOENV, FLOOREC e LOGBM 146
Figura 50 Rede Básica Usada no Teste Prático.....147
Figura 51 Rede Completa Usada no Teste Prático 148
Figura 52 Exemplo de Arquivo de Log.....150
Figura 53 Tempo médio para mensagens em rede com 9 X 167 micros.....157
Figura 54 Tempo médio para mensagens *Reliable* X *Safe* 158
Figura 55 Tempo médio para mensagens de 100 X 1000 *bytes* 159

Lista de Quadros

Quadro 1 Características de Sistemas Distribuídos 23
Quadro 2 Características Principais dos Computadores Paralelos 27
Quadro 3 Mecanismos de Comunicação em Grupo e Ambientes de Rede 81
Quadro 4 Número de Máquinas por Servidor SPREAD 149
Quadro 5 Número de Máquinas por Local Físico 149
Quadro 6 Medidas Rede Simples Mensagens 1000 Reliable 152
Quadro 7 Medidas Rede Simples Mensagem 1000 Safe 152
Quadro 8 Medidas Rede Simples Mensagem 100 Reliable 153
Quadro 9 Medidas Rede Simples Mensagem 100 Safe 153
Quadro 10 Medidas Rede Completa Mensagem 1000 Reliable 154
Quadro 11 Medidas Rede Completa Mensagem 1000 Safe 154
Quadro 12 Medidas Rede Completa Mensagem 100 Reliable 155
Quadro 13 Medidas Rede Completa Mensagem 100 Safe 155

Lista de Abreviaturas e Siglas

ACK	*Acknowledgement*
ANSI	*American National Standards Institute*
API	*Application Program Interface*
ARPANET	*Advanced Research Projects Agency Network*
ASCII	*American (National) Standard Code fo r Information Interchange*
ATM	*Asynchronous Transfer Mode*
CC-NUMA	*Cache-Coherente Nonuniform Memory Access*
CEJURPS	Centro de Educação Superior de Ciências Jurídicas, Políticas e Sociais
CESCIESA	Centro de Educação Superior de Ciências Sociais Aplicadas
COM/OLE	*Common Object Model / Object Linking and*
CORBA	*Common Object Request Broker Architecture*
CTTMar	Centro de Educação de Ciências Tecnológicas da Terra e do Mar
DARP	*Defense Advanced Research Project Agency*
DCE	*Distributed Computing Environment*
DCOM	*Distributed Component Object Model*
DIL	*Daemon Interaction Layer*
DNS	*Domain Name System*
EBCDIC	*Extended Binary Coded Decimal Interchange Code*
FIFO	*First In First Out*
GCC	*GNU Compiler Collection*
GM	*Group Manager*
GMI	*Group Method Invocation*
GMIL	*Group Method Invocation Layer*
GMIS	*Group Method Invocation Service*
HTTP	*Hiper Text Transfer Protocol*
IDL	*Interface Definition Language*
IEEE	*Institute of Electrical and Electronics Engineers*
IGRM	*Inter Group Reliable Multicast*
IGTM	*Inter Group Thread ed Module*
IP	*Internet Protocol*
IPX	*Internetwork Packet Exchange*
ISSO	*International Standard Organization*
JDK	*Java Development Kit*
JVM	*Java Virtual Machine*

LAN	*Local Area Network*
LTS	*Lamport Time Stamp*
MIMD	*Multiple Instruction Stream Multiple Data Stream*
MP	*Message Publisher*
MPI	*Message Passing Interface*
MPP	*Massively Parallel Processors*
MS	*Message Subscriber*
MSL	*Multi Send Layer*
NACK	*No Acknowledgement*
NFS	*Network File System*
OG	*Open Group*
OMA	*Object Management Architecture*
OMG	*Object Management Group*
ORB	*Object Request Broker*
OSI	*Open Systems Interconnection*
PC	*Personal Computer*
PGML	*Partitionable Group Membership Layer*
PGMS	*Partitionable Group Membership Service*
PVM	*Parallel Virtual Machine*
RMI	*Remote Method Invocation*
RML	*Reliable Multicast Layer*
RMP	*Reliable Multicast Protocol*
ROT	*Running Object Table*
RPC	*Remote Procedure Call*
SCI	*Scalable Coherente Interface*
SEPLAN	Secretaria de Planejamento
SIMD	*Single Instruction Stream Multiple Data Stream*
SISD	*Single Instruction Stream Single Data Stream*
SML	*State Merging Layer*
SMP	*Symmetric Multiprocessors*
SMS	*State Merging Service*
SPX	*Sequenced Packet Exchange*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol / Internet Protocol*
UDP	*User Datagram Protocol*

UNIVALI	Universidade do Vale do Itajai
URL	*Universal Resource Locator*
WAN	*Wide Area Network*

1 INTRODUÇÃO

Redes de computadores estão por toda parte. A Interne é uma, como são as muitas redes que a compõem. Re des de telefonia celular, redes corporativas, redes fabris, redes em campus, redes domésticas, redes embarcadas, todas estas sejam redes locais LAN (*Local Area Network*) ou redes de longa distância WAN (*Wide Area Network*), tanto separadamente como em conjunto, compartilham as características essenciais que fazem delas um modelo relevante para o estudo sob o título Sistemas Distribuídos.

Uma definição de sistemas distribuídos é aquela na qual os componentes de hardware e software localizados em computadores interligados por rede, comunicam e coordenam suas ações somente através da troca de mensagens (COULOURIS, 2001).

A comunicação entre os computadores interligados, pode ocorrer de diferentes formas, usando diversos protocolos tanto em ambiente de rede local como ambiente de redes de longa distância. Um caso particular de comunicação ocorre quando uma mesma mensagem tem que ser enviada para diversos computadores na rede. Alguns ambientes físicos de rede permitem o *broadcast*, onde um único comando de envio manda a mensagem para todos os endereços da rede, porém a confiabilidade deste método não é garantida, podendo ocorrer alguma perda sem que o emissor da mensagem seja notificado da mesma.

Com o intuito de atender a necessidade de comunicação de um para muitos, com confiabilidade, escalabilidade, e de forma transparente para quem desenvolve a aplicação, foi criado um novo paradigma chamado Comunicação em Grupo. Um protocolo de Comunicação em Grupo separa a complexidade de controle e gerenciamento das mensagens, da complexidade da aplicação, tratando os diversos computadores como sendo pertencentes a um grupo. Assim, o projetista se preocupa apenas com a aplicação, suas funcionalidades, seus procedimentos, seus algoritmos particulares, sem envolver-se com os problemas da comunicação. Passa a ser responsabilidade do Protocolo de Comunicação em Grupo a controle de fluxo, a confiabilidade, o sequenciamento e a garantia de entrega das mensagens ao aplicativo final, bem como, o controle de quais computadores estão faze ndo parte deste grupo, gerenciando a entrada de novos membros no grupo, saída voluntária ou saída involuntária devido à quebra no computador ou particionamento da rede. Um

particionamento da rede acontece quando existe uma separação de dois segmentos de redes que perdem a comunicação entre si.

Ao considerar a importância dos mecanismos de Comunicação em Grupo existentes, este trabalho visa realizar uma avaliação nas diferentes alternativas disponíveis e em especial naquelas que tratam ambientes de redes de longa distância (WAN), onde o particionamento, a latência e o assincronismo tornam este paradigma bem mais complexo.

Para ilustrar de forma prática o uso de um mecanismo de Comunicação em Grupo, será implementada uma aplicação que visa o gerenciamento de recursos em sistemas distribuídos.

1.1 Objetivos

Este trabalho tem como objetivo geral fazer uma avaliação dos mecanismos de Comunicação em Grupo para ambiente WAN e como objetivos específicos pode-se citar:

- Estudar sistemas distribuídos e paralelos.
- Estudar as características de hardware que atendem sistemas distribuídos.
- Estudar as características dos softwares que atendem sistemas distribuídos.
- Estudar os *midleware* (camadas de software entre os aplicativos e o sistema operacional) de sistemas distribuídos, CORBA, JAVA RMI, DCOM, DCE.
- Estudar detalhadamente Comunicação em Grupo.
- Avaliar conceitualmente os principais mecanismos de Comunicação em Grupo.
- Selecionar três mecanismos para um estudo mais detalhado.
- Selecionar um dos três para usar em uma implementação prática.
- Desenvolver uma aplicação para gerenciamento de recursos em um ambiente de sistemas distribuídos.
- Fazer uma avaliação de performance, do mecanismo selecionado, considerando volume, escalabilidade e funcionalidades.

1.2 Motivação

O uso de mecanismos de comunicação em grupo, torna a aplicação muito simples de ser codificada, pois toda a complexidade da comunicação, seus controles, sua segurança, seu fluxo, passam a ser tratados de forma transparente para quem faz a aplicação. Também o fator escalabilidade é muito importante, pois o aplicativo que roda para um computador, poderá rodar para N computadores, sem nenhuma alteração.

No passado, quando tinha-se apenas sistemas centralizados com o uso de terminais, o gerenciamento dos recursos eram simples pois, todos os recursos a serem mensurados estavam centralizados em um único local.

Com o ambiente de redes, onde tem-se eventualmente que administrar milhares de microcomputadores, a gerência destes recursos se torna um problema complexo devido a localização e grande quantidade de máquinas a serem consideradas no gerenciamento.

Assim, a idéia é aproveitar todas as facilidades que um mecanismo de comunicação em grupo oferece, e desenvolver uma aplicação cliente que estara rodando em todas as máquinas, fazendo parte de um grupo geral, o qual fornece informações de gerenciamento a um ponto central de gerenciamento da rede, caracterizando assim uma aplicação de gerência de recursos em sistemas distribuídos.

1.3 Materiais e Métodos

O mecanismo de comunicação em grupo selecionado para ser usado na implementação de uma aplicação prática foi o SPREAD. Assim, configura-se a rede para que em cada segmento da mesma tenha-se um *daemon*, uma cópia do SPREAD executando e todos os clientes locais deste segmento, sendo controlados por esta cópia local. Como se tem acesso ao fonte do SPREAD, e já estando portado para múltiplas plataformas como diversos UNIX, MAC e WINDOWS, efetua-se a compilação em LINUX. Optou-se por rodar o SPREAD em um servidor usando LINUX, o mesmo que hoje já é usado para os serviços de WEB e E-mail nestes segmentos de redes.

Para os aplicativos foram usados ambientes WINDOWS com compilador Visual C e LINUX com compilador GCC. As aplicações em ambiente gráfico, clientes que rodam em máquinas WINDOWS, e programa de consulta com interface gráfica, foram desenvolvidos em C e compilados com Visual C da Microsoft. Já o programa que coleta

as informações da hora que a máquina foi ligada ou desligada, gravando um *log* (arquivo com registros de controle gravados em ordem cronológica) em disco, foi desenvolvido em C, compilado com GCC no LINUX e roda no mesmo servidor que executa o SPREAD.

Para os aplicativos que medem performance, avaliando o tempo no recebimento de diferentes conjuntos de mensagens, foram empregados compiladores Visual C e GCC sendo que todos os programas rodam tanto em WINDOWS como em LINUX.

Como ambiente de redes, usou-se a topologia da UNIVALI, com redes locais, servidores locais em cada um dos Campi, usando *fast ethernet* e *gigabit ethernet* sendo os pontos de rede geralmente ligados a *switch*. Os segmentos locais estão interligados ao Campus principal de Itajaí por *links* WAN de 2 Megabits.

1.4 Conteúdo do Documento

O presente documento contém inicialmente o capítulo 2 onde é apresentada a conceituação básica para sistemas distribuídos. Além das características relativas ao hardware e os softwares básicos que atendem o conceito de sistemas distribuídos, também o paradigma de orientação a objeto é abordado. Assim como, o uso de CORBA, JAVA RMI, DCOM e DCE são considerados nesta parte de conceituação básica.

O capítulo 3, é inteiramente dedicado para detalhar conceitualmente o que é Comunicação em Grupo, com suas características, funcionalidades, requisitos e padrões adotados.

No capítulo 4, tem-se uma visão geral de diversos mecanismos de Comunicação em Grupo, com suas características principais. Como foram selecionados três ambientes para um estudo mais detalhado, cada um deles passa a ser detalhado em um capítulo específico, sendo no capítulo 5 o INTERGROUP, no capítulo 6 o JGROUP e no capítulo 7 o SPREAD.

A descrição da aplicação de gerência de recursos em ambiente distribuído, bem como os resultados da avaliação de desempenho estão detalhados no capítulo 8.

Para finalizar tem-se a conclusão no capítulo 9 e na seqüência os anexos com os programas fontes que foram desenvolvidos para avaliação de desempenho usando o SPREAD.

2 CONCEITUAÇÃO BÁSICA

Neste capitulo, apresenta-se uma série de conceitos que são importantes para o perfeito entendimento e enquadramento do assunto objeto, Comunicação em Grupo. Inicia-se com uma caracterização dos sistemas distribuídos e paralelos, bem como, apresentando motivações para o uso de sistemas distribuídos. Na seqüência é abordado o aspecto de hardware, apresentando arquiteturas atualmente em uso para sistemas distribuídos. Os aspectos de software como sistemas operacionais, linguagens de programação e ambientes distribuídos também são explorados neste capítulo. Para encerrar a conceituação básica, apresentam-se as arquiteturas adotadas na comunicação de sistemas distribuídos.

2.1 Características dos Sistemas Distribuídos e Paralelos

Os primeiros computadores produzidos para uso comercial, na década de 70, tinham uma estrutura padrão baseada em processador, memória e armazenamento de dados. Devido ao alto custo dos componentes, na época, as capacidades de memória e de discos destes precursores dos equipamentos hoje conhecidos como *mainframe*, eram mínimas se comparado com equipamentos atuais. Para exemplificar um modelo de equipamento IBM 360/30, tinha memória de 32Kb, 3 discos de 10 MB, leitora perfuradora de cartões, leitora de fita de papel, e 6 unidades de fitas magnéticas. Na evolução tem-se os minicomputadores, os microcomputadores, o computador pessoal ou PC e a interligação dos equipamentos em rede.

A interligação das máquinas em redes propiciou o surgimento do conceito de sistemas distribuídos, aproveitando o potencial das máquinas interligadas. De outro lado, tem-se os sistemas compostos de um único processador, memória e periféricos os quais são chamados de sistemas centralizados. Segundo Tanenbaum (1995), no passado era válido a Lei de Grosch, um especialista em computação, preconizando que “o poder computacional de um processador é proporcional ao quadrado de seu preço, ou seja, pagando duas vezes mais, pode-se obter o quádruplo da performance”. Hoje em dia esta Lei de Grosch não é mais válida, pois com as novas tecnologias, em especial usando

arquitetura INTEL a cada ano, pelo mesmo preço, obtemos o dobro ou o quádruplo de performance em relação ao equipamento anterior.

Quando se avalia o custo de micros pessoais PC, encontra-se ótima relação preço/performance. Quando se coloca uma máquina, independente para cada usuário, tem-se a dificuldade de compartilhar os dados. Além da comunicação necessária entre os usuários, é desejável o compartilhamento de recursos tais como impressoras, gravador de CD, etc. Este compartilhamento é possível se as máquinas estiverem interligadas em rede. Assim um sistema distribuído pode ser implementado com computadores pessoais ligados em rede a servidores, os quais executariam os processos mais pesados de forma distribuída.

Apesar de todas as vantagens dos sistemas distribuídos, encontram-se algumas dificuldades neste ambiente. O software para sistemas distribuídos é mais complexo. Desta forma sistemas operacionais, linguagens e mesmo aplicativos para este ambiente são relativamente novos, em fase de pesquisa, e não estão plenamente consolidados. Também a rede para interligar, seja rede local (LAN- *Local Area Network*) ou rede de longa distância (WAN- *Wide Area Network*), não tem a confiabilidade total para a interligação, podendo ocorrer perda por interrupção do *link* (caminho que interliga fisicamente dois pontos de rede) ou saturação devido à sobrecarga na banda disponível. Cabe então as soluções de software, tratar estas interrupções com protocolos e rotinas adequadas a este contexto. Um resumo das características de sistemas distribuídos, é mostrado no Quadro 1 segundo Tanenbaum (1995).

2.2 Estratégias para Sistemas Distribuídos

As estratégias adotadas para a implementação de sistemas distribuídos dependem de diversos fatores. Tem-se que levar em conta o objetivo da aplicação, as características e restrições do ambiente, o custo do projeto, etc. As estratégias mais importantes, segundo Veríssimo (2001), são: distribuição para informação e compartilhamento de recursos; distribuição para maior disponibilidade e performance; distribuição para modularidade; distribuição para a descentralização e distribuição para segurança.

Quadro 1 Características de Sistemas Distribuídos

ITEM	DESCRIÇÃO
Relação Custo Benefício	Os microprocessadores oferecem uma melhor relação preço/performance do que a oferecida pelos *mainframes.*
Desempenho	Um sistema distribuído pode ter um poder de processamento maior que o de qualquer *mainframe*.
Confiabilidade	Se uma máquina sair do ar, o sistema como um todo pode sobreviver
Crescimento Incremental	O poder computacional pode crescer em doses homeopáticas
Compartilhamento	Permite que mais de um usuário acesse uma base de dados comum, ou periféricos muito caros.
Comunicação	Torna muito mais simples a comunicação pessoa a pessoa, por exemplo, empregando o correio eletrônico.
Flexibilidade	Espalha a carga de trabalho por todas as máquinas disponíveis ao longo da rede.
Restrições	Até o presente momento não há muita disponibilidade de software para os sistemas distribuídos. Pode-se ter problemas de muito tráfego na rede ou problemas com segurança dos dados.

2.2 Estratégias para Sistemas Distribuídos

As estratégias adotadas para a implementação de sistemas distribuídos dependem de diversos fatores. Tem-se que levar em conta o objetivo da aplicação, as características e restrições do ambiente, o custo do projeto, etc. As estratégias mais importantes, segundo Veríssimo (2001), são: distribuição para informação e compartilhamento de recursos; distribuição para maior disponibilidade e performance; distribuição para modularidade; distribuição para a descentralização e distribuição para segurança.

2.2.1 Considerações Básicas

Alguns compromissos precisam ser atendidos, dependendo das particularidades das aplicações:

- **Controle centralizado versus descentralizado** – Controle centralizado é mais fácil de implementar e gerenciar. Contudo, alguns problemas de descentralização e natureza distribuída, são melhores resolvidos com aplicações descentralizadas. Também podem ser consideradas arquiteturas intermediárias com transparência distribuída, mas de controle centralizado.
- **Distribuição seqüencial versus concorrente** - O uso do processamento concorrente, onde tem-se diversos procedimentos executando ao mesmo tempo devido a requisições simultâneas, comparativamente ao processamento seqüencial no caso de chamadas que podem ser atendidas uma a uma.
- **Distribuição visível versus invisível** – Pode-se dizer que a distribuição é invisível ou transparente quando se usam modelos que escondem a distribuição. Assim em modelos usando RPC (*Remote Procedure Call*), CORBA (*Common Object Request Broker Architecture*) ou DCOM (*Distributed Component Object Model*) a distribuição se torna transparente. Distribuição visível é quando a aplicação se vale de mecanismos de passagem de mensagens, tais como orientados a grupos e modelos usando barramento de mensagens.
- **Envio de dados versus código** – O que é distribuído, dados ou código? Normalmente distribuem-se dados para serem processados em equipamentos diferentes, porém, é possível também, enviar código para processar as instruções localmente, como no caso de *applets.*
- **Servidor versus serviço** – Deve-se considerar que um serviço pode estar em mais do que um servidor e que em um servidor pode-se estar executando diversos serviços.
- **Escala versus performance** – A escalabilidade normalmente é inversamente proporcional à performance. Uma boa arquitetura que permita escalabilidade, ou seja, possa crescer em número de componentes, não deve ter a sua performance linearmente diminuída, devido a este crescimento.

- **Sincronismo versus assincronismo** – Assíncronos são sistemas mais simples, mas independentes de tempo, enquanto síncronos são sistemas mais complexos e tem a habilidade de assegurar especificações de controle de tempo.

Muitos destes compromissos podem ser avaliados em conjunto e assim pode-se classificar a distribuição como segue:

- **Distribuição em baixa escala** – Em ambientes mais homogêneos usando LANs ou redes de alta velocidade, onde um comportamento controlado pode ser obtido para aplicações que requerem segurança e sincronismo
- **Distribuição em larga escala** – Em ambientes de sistema distribuídos, em ambientes abertos, tipicamente de natureza heterogênea, sobre WANs interconectando LANs onde o comportamento é incerto e aplicações podem existir de forma não síncrona.

2.2.2 Distribuição para Informação e Compartilhamento de Recursos

Esta é a forma como nasceram sistemas distribuídos e ainda é o objetivo estratégico da maior parte dos sistemas distribuídos. A tecnologia que diz respeito a esta estratégia de implementação, está relacionada com servidores centrais, protocolos de sessão remota, computação cliente-servidor, HTTP, cliente leve e computação em rede. Também está relacionada com as clássicas formas de disseminação da informação como NEWS, BBS e *e-mail*. Os esforços estão concentrados no lado do usuário, para garantir o acesso às informações e recursos que residem em servidores centrais. São importantes neste contexto o gerenciamento de usuários, segurança no estabelecimento da sessão, largura de banda e confiabilidade e software no lado cliente.

2.2.3 Distribuição para Maior Disponibilidade e Performance

Na estratégia de distribuir para maior disponibilidade e performance, o que se tem é a vantagem de diversos servidores contra um único servidor central ou mesmo *mainframe*. Se se tem um único ponto de falha com servidor central, com múltiplos servidores tem-se uma disponibilidade muito maior, pois em caso de queda de um dos

servidores, ainda tem se a disponibilidade de atendimento por outros servidores. Quanto à performance, um único equipamento tem um limite na sua performance, enquanto que com diversos servidores tem se a possibilidade de balancear o acesso aos servidores, o que nos dará uma performance muito superior a de um único servidor central.

2.2.4 Distribuição para Modularidade

Mesmo para uma empresa que tenha todas as facilidades de computação centralizadas, onde todos os usuários estão no mesmo ambiente, ainda assim, a modularidade é relevante como uma das estratégias. Mesmo com um custo superior se comparado a um único sistema integrado, na modularidade podemos ter benefícios no gerenciamento das incertezas quanto ao crescimento da organização e sua atividade (geografia, escala, reengenharia, reorientação, mercados, etc). Em atendendo a modularidade, adicionalmente têm-se as vantagens discutidas anteriormente de disponibilidade e performance.

2.2.5 Distribuição para a Descentralização

Diversas atividades conduzidas por pessoas nas organizações, são descentralizadas pela natureza da própria atividade. Antes dos sistemas distribuídos, ou se tinha esta facilidade de computação em um computador central, ligando terminais remotos aos locais do usuário ou se tinha ilhas de processamento com computadores isolados atendendo estas aplicações descentralizadas. Com a estratégia de distribuir para a descentralização, passa-se a ter integração e ao mesmo tempo a disponibilidade em áreas descentralizadas das empresas. Esta estratégia permite descentralizar o controle, colocando ele onde for necessário, enquanto mantém o grau de integração e coordenação entre os diferentes pontos distribuídos.

2.2.6 Distribuição para Segurança.

A segurança pode ser aumentada quando se tem distribuição. Em áreas de proteção e criptografia baseadas em servidores de votação, tem se mais robustez se

comparado a sistemas de um único servidor. Sistemas de arquivamento baseados em fragmentação e espalhamento em diversos servidores de arquivos, aumentam a segurança se comparado a servidor único que contém o arquivo inteiro.

2.3 Classificação de Sistemas Distribuídos e Paralelos

Segundo Buyya (1999), "a principal razão para a criação e o uso de computadores paralelos é que o paralelismo é uma das melhores formas de resolver o problema de gargalo de processamento em sistemas de um único processador". A relação preço performance de um pequeno sistema de processamento paralelo em cluster é muito melhor se comparado com um minicomputador.

Quadro 2 Características Principais dos Computadores Paralelos

Características	MPP	SMP CC-NUMA	*Cluster*	Distribuição
Número de Nós	0(100)- (1000)	0(10)-0(100)	0(100)ou menos	0(10)-0(1000)
Complexidade do Nó	Granularidade fina ou média	Granularidade Média ou grossa	Granularidade média	Alcançe largo
Comunicação inter Nos	Troca mensagens / variáveis compartilhadas	Centralizada e Memória compartilhada distribuída	Troca de mensagens	Arquivos compartilhados, RPC , IPC e troca mensagens
Controle de *jobs*	Fila única no host	Fila única geralmente	Múltiplas filas mas coordenadas	Filas independentes
Suporte a SSI Imagem de Sistema Único	Parcialmente	Sempre no SMP e alguns NUMA	Desejável	Não
Copias de SO e tipo	N micro-kernel monolítico ou SO em camadas	Um monolítico SMP e muitos para NUMA	N SO plataforma homogênea ou micro-kernel	N SO plataforma homogêneas
Espaço endereçamento	Múltiplo – Simples para DSM	Simples	Múltiplo ou Simples	Múltiplo
Segurança entre os nós	Não necessário	Não Necessário	Requer se exposto	Requer
Onde se aplica	Uma Empresa	Uma Empresa	Uma ou mais empresas	Muitas empresas

O desenvolvimento e a produção de sistemas de moderada velocidade usando arquitetura paralela é muito mais barato do que o seu equivalente em performance para sistemas de um único processador. A taxonomia das arquiteturas de computadores

paralelos, de acordo com seus processadores, memória e forma de interconexão pode ser (BUYYA, 1999):

- MPP (*Massively Parallel Processors*) Processadores Paralelos
- SMP (*Symmetric Multiprocessors*) Multiprocessadores simétricos
- CC-NUMA (*Cache-Coherente Nonuniform Memory Access*) Processador com memória cache e que também acessa as memórias dos outros processadores
- Sistemas Distribuídos
- *Cluster*

O Quadro 2 mostra uma comparação das características funcionais e arquitetura destas máquinas.

A classificação de Buyya (1999), segue a classificação de Tanenbaum (1995), que é mais geral e que refina a classificação apresentada por Flynn apud Tanenbaum (1995), para arquitetura de computadores. A taxonomia de Flynn leva em consideração fluxo de instrução e fluxo de dados, como segue:

- SISD – *Single Instruction Stream, Single Data Stream* – Um único fluxo de instruções e um único fluxo de dados.
- SIMD – *Single Instruction Stream, Multiple Data Stream* – Um único fluxo de instruções e múltiplo fluxo de dados.
- MIMD – *Multiple Instruction Stream, Multiple Data Stream* – Múltiplo fluxo de instruções e múltiplo fluxo de dados.

Para Tanebaum (1995), as máquinas com múltiplo fluxo de instruções e múltiplo fluxo de dados ainda se dividem nos chamados **multiprocessadores,** para os que têm memória compartilhada e **multicomputadores** para os que não tem memória compartilhada. Tanto os multiprocessadores como os multicomputadores podem ser divididos em duas categorias, tomando por base a arquitetura de rede de interconexão, sendo uma do tipo baseada em barramento e outra comutada respectivamente. Os multiprocessadores cuja conexão é feita com barramento, ou seja um *backplane* de alta velocidade, são também chamados de **fortemente acoplados** e os multicomputadores interligados via rede ou outros dispositivos comutados em baixa velocidade, são também chamados de **fracamente acoplados**. A Figura 1 (TANENBAUM, 1995), mostra esta classificação para os sistemas distribuídos e paralelos.

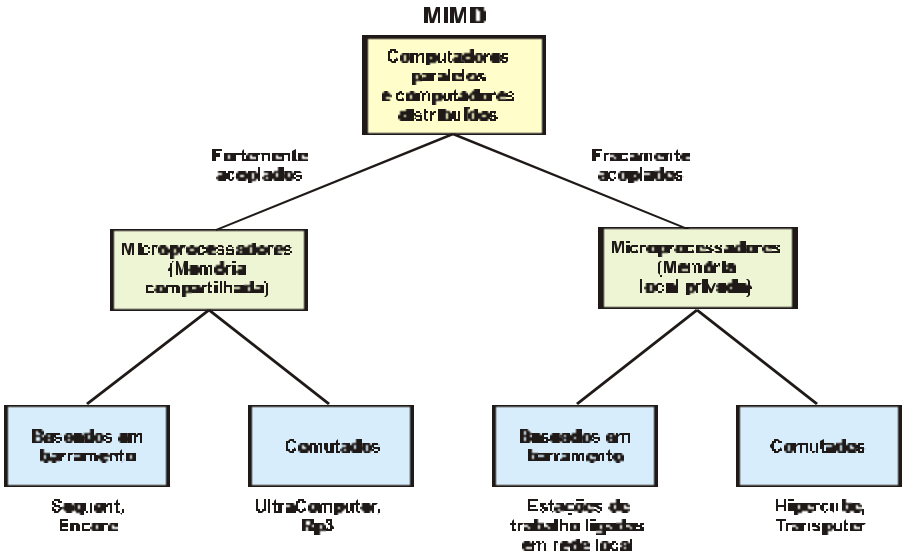

Figura 1 Uma Taxonomia para Sistemas Distribuídos e Sistemas Paralelos

2.3.1 Multiprocessadores

A característica básica dos multiprocessadores é que a comunicação entre os processadores se dá através de memória compartilhada. Uma área única de memória sendo acessada simultaneamente por diversos processadores pode causar um gargalo. Assim, o conceito de memória de alta velocidade de acesso também conhecido como memória *cache*, passa a ser adotado, tendo cada processador a sua memória *cache* própria. Para manter todos os *cache* atualizados uma técnica chamada *write-trhough* onde toda a gravação é feita na memória principal e a leitura na memória *cache*. Também todas as *cache* ficam monitorando o barramento e se uma alteração ocorre na memória principal cujo endereço também está na *cache* de um processador este atualiza o conteúdo da *cache*. A Figura 2 (TANENBAUM, 1995), ilustra um sistema distribuído baseado em barramento.

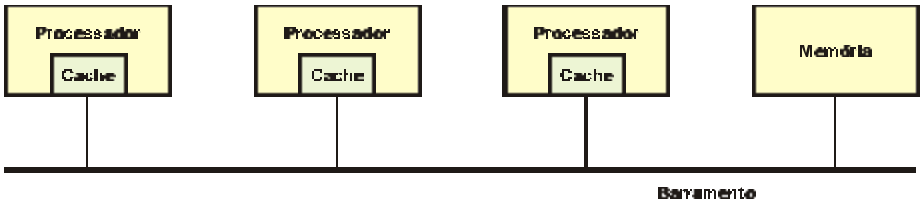

Figura 2 Sistema Distribuído Baseado em Barramento

Os sistemas multiprocessadores podem também ser ligado por comutação. Assim divide-se a memória em blocos e o acesso dos processadores aos blocos de memória pode ser feito através de comutação *crossbar* como mostra a Figura 3 (TANENBAUM, 1995). Em cada intersecção existe uma chave eletrônica que pode ser aberta ou fechada por hardware.

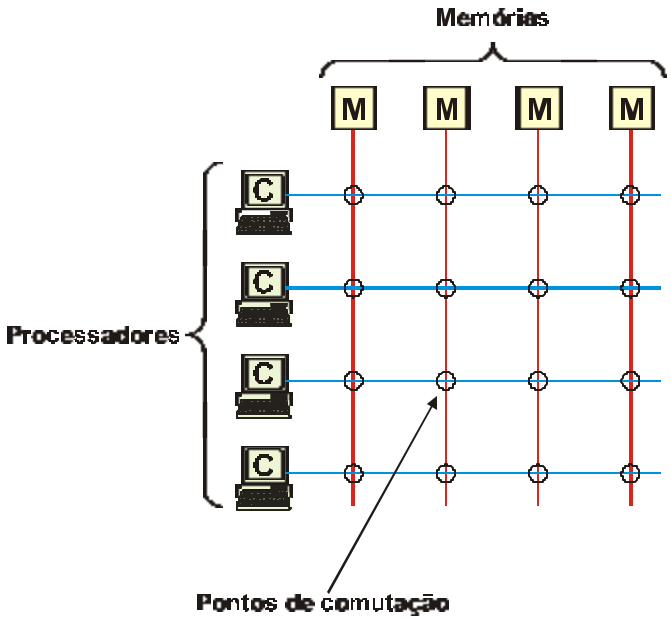

Figura 3 Comutação Crossbar

Alternativa é usar uma solução denominada rede Omega, onde cada chave tem duas entrada e duas saídas reduzindo o número de comutações de n^2 para nlog2n. A Figura 4 (TANENBAUM, 1995), mostra esta alternativa para n igual a 4.

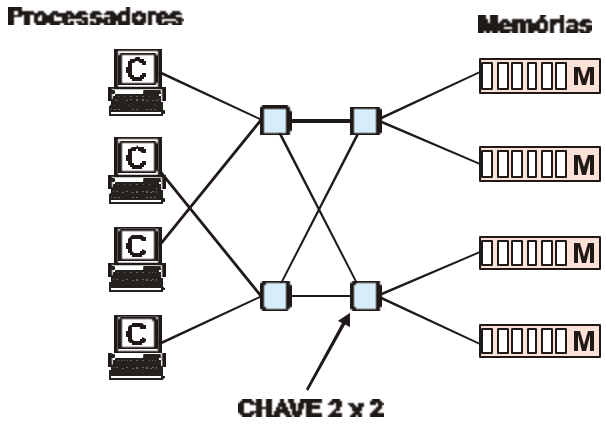

Figura 4 Rede Omega

Esta alternativa exige chaves de comutação muito rápidas e se tivermos um número de processadores muito grande as chaves de comutação se tornam um gargalo.

2.3.2 Multicomputador

Os sistemas multicomputadores, onde não se tem memória compartilhada entre os processadores, não apresentam as dificuldades citadas anteriormente. Para os sistemas multicomputadores ligados em barramento a interconexão pode ser feita via rede local onde a velocidade de placas de 100 a 1000 Mbps é suficiente para o tráfego de comunicação processador-a-processador. Esta conexão é suportada inclusive entre redes de maior distância com largura de banda adequada, usando tecnologia como ATM ou Gigabit Ethernet. Na Figura 5 (TANENBAUM, 1995), pode-se ver um conjunto de estações de trabalho ligada em rede que ilustram este sistema multicomputador baseado em barramento.

Figura 5 Sistema Constituído por Computadores Interligados em Rede

Outra forma de ligação de multicomputadores é a interligação por comutação. Duas topologias são apresentadas por Tanenbaum (1995), sendo a primeira chamada de topologia em grade, facilmente implementada em placas de circuito impresso, e mostrada na Figura 6.

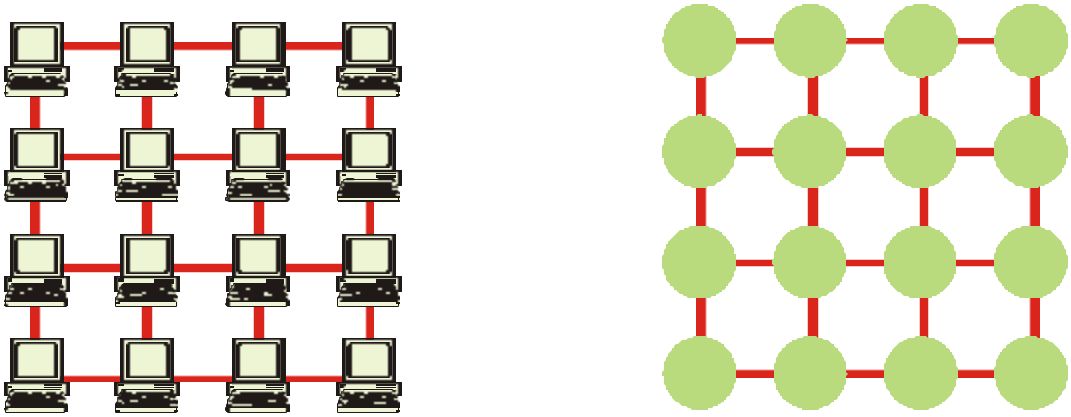

Figura 6 Topologia em Grade

A segunda topologia é chamada de hipercubo, onde os vértices representam os processadores e os lados às interligações entre os processadores. Na Figura 7 (TANENBAUM, 1995), uma representação para este esquema com dimensão 4 onde cada processador está ligado a outros 4 processadores.

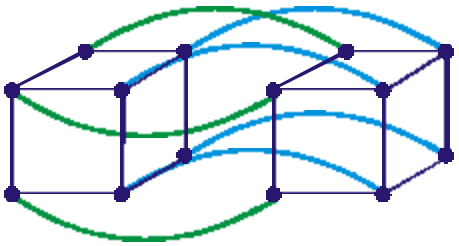

Figura 7 Topologia em Hipercubo

Nesta forma de hipercubo à medida que se aumenta o número de processadores maior fica o caminho para os processadores se comunicarem entre si, pois a interligação não é direta e sim via os processadores que são vizinhos na interligação.

2.3.3 Cluster

Segundo Buyya (1999), “cluster é um tipo de sistema de processamento distribuído ou paralelo, o qual consiste de um conjunto de computadores interligados trabalhando juntos como se fosse um único recurso de computação integrada”. Um nó pode ser um computador multiprocessado ou monoprocessado (PC, estação de trabalho ou SMP) com memória, facilidade de I/O e sistema operacional. Um *cluster* geralmente se refere a dois ou mais computadores (nós) conectados entre si. Os nós podem estar em um único gabinete ou interligados via LAN. A Figura 8 mostra uma arquitetura típica de um cluster (BUYYA, 1999).

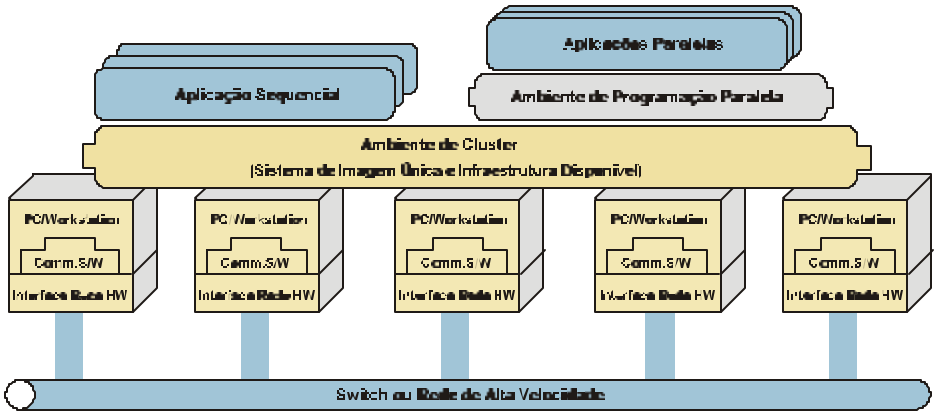

Figura 8 Arquitetura Típica de um Cluster

Cluster oferece as seguintes facilidades a um custo relativamente baixo:

- Alta performance
- Escalabilidade e facilidade de expansão
- Alta performance de processamento
- Alta disponibilidade

A tecnologia de *cluster* permite aumentar o poder de processamento, usando tecnologia comum a preços relativamente baixos. Isto devido ao uso de hardware e software, que são padrões de mercado e com preços muito abaixo dos convencionais então existentes. Um benefício muito importante quando se usa *cluster* é a capacidade de suportar falhas, pois, se um nó do *cluster* deixa de funcionar, o sistema como um

todo ainda continua funcionando, apenas com uma performance inferior devido ao nó que deixou de processar.

A interconexão entre os nós de um sistema em *cluster* pode ser feita usando redes de alta velocidade com protocolo standard como TCP/IP ou protocolos de baixo nível como Mensagens Ativas. O mais comum é que se use placa de rede do tipo Ethernet. Em termos de performance, latência e largura de banda, esta tecnologia está evoluindo. Uma conexão Ethernet simples não pode ser usada como base para uma ligação *cluster*, pois a latência e a largura de banda não estão de acordo com o poder computacional que temos nas atuais estações de trabalho. Deve-se esperar que na interconexão de cluster tenha-se largura de banda superior a 10 Mbytes/s e tempo de latência menor que 100 ìs. Algumas das tecnologias de rede de alta performance, segundo Buyya (1999), são apresentadas a seguir:

- ***Ethernet, Fast Ethernet e Gigabit Ethernet*** - A *ethernet* tem se tornado um padrão em redes de estações de trabalho. Inicialmente tínhamos as placas de 10 Mbps, seguidas pelas placas *fast ethernet* com velocidade 100 Mbps usando *hubs* ou *switch* e atualmente, o estado da arte, é a placa *gigabit ethernet* que mantêm a mesma simplicidade da arquitetura *ethernet* permitindo velocidades da ordem de *gigabit* por segundo. Pode-se também agregar múltiplas placas, formando um fluxo único com a transmissão de diversas placas simultaneamente, chamado *trunking*, e tem-se uma interconexão de alta velocidade entre as estações da rede.
- ***Asynchronous Transfer Mode*** **(ATM)** - ATM é uma tecnologia de circuito virtual chaveado, originalmente desenvolvida para a indústria de telecomunicações. Baseado na transmissão de pequenos pacotes de dados chamados células pode ser usado tanto em LAN como WAN. As placas com tecnologia ATM têm um preço muito caro e não são largamente adotadas.
- ***Scalable Coherente Interface*** **(SCI)** - O objetivo do padrão SCI, ANSI/IEEE 1596-1992, é essencialmente prover um mecanismo de alta performance que suporte o acesso à memória compartilhada coerente, entre um grande número de máquinas. Esta transferência é feita com um tempo de espera (*delay*) de poucos ìss. Esta tecnologia não é largamente empregada, seu custo é bastante alto e podemos encontrar em algumas arquiteturas proprietárias de memória logicamente compartilhada e

fisicamente distribuída, tais como HP/Convex Exemplar SPP e Sequent NUMA-Q 2000 (DIETZ, 1998).

- ***Myrinet*** - É uma solução proprietária da empresa Myricom (2001), de alta performance para interconexão a 1.28 Gbps full duplex. Myrinet usa *switch* roteadores *cut-through* de baixa latência, oferecendo tolerância à falha pelo mapeamento automático da configuração de rede. Tem baixo tempo de latência (5 ìs sentido único, ponto a ponto), excelente performance e grande flexibilidade devido ao processador programável em sua placa.

Outras tecnologias de rede e formas de interligação de computadores tais como, ArcNet, CAPERS, FC (*fibre channel*), FireWire (IEEE 1394), HiPPI e serial HiPPI, IrDA (*Infrared Data Association*), Parastation, PLIP, SCSI, ServNet, SHRIMP, SLIP, TTL_PAPERS, USB e WAPERS estão descritas em (DIETZ, 1998).

2.4 Software em Sistemas Distribuídos

O software em Sistemas Distribuídos seja aplicativo ou mesmo um sistema operacional, tem que estar de acordo com as características dos Sistemas Distribuídos. Diferentemente de um sistema monoprocessado no ambiente de Sistemas Distribuídos existem as necessidades de sincronismo, interação, controles, entre os diferentes ambientes de software, ou diferentes instâncias, que compõem o sistema total. Para o usuário final, tudo deve parecer como se fosse um único recurso de computação que está sendo usado para o processamento das necessidades do usuário. Neste contexto, vamos fazer a seguir, algumas considerações sobre os sistemas operacionais e sobre a programação de aplicativos.

2.4.1 Sistemas Operacionais

Segundo Buyya (1999), um sistema operacional moderno provê dois serviços fundamentais para o usuário. Primeiro ele permite utilizar o hardware de um computador mais facilmente criando uma máquina virtual que difere da máquina real facilitando o uso da mesma por usuários finais. Segundo, um sistema operacional compartilha recursos de hardware entre os usuários. Um dos mais importantes recursos

é o processador. Um sistema operacional que trabalha com mais de um processo simultaneamente (*multitask*), como o Unix ou WINDOWS NT, divide o trabalho que necessita ser executado pelo processador, dando a cada processo memória, recursos do sistema e uma fração de tempo do processador. O sistema operacional executa um processo (*thread*) por um curto espaço de tempo e depois muda para outro, executando um a um sucessivamente. Assim mesmo em um sistema com um único processador, temos a impressão de múltiplas execuções simultâneas, pois cada processo está sendo atendido pelo processador em pequenas frações de tempo. Um usuário pode editar um documento, enquanto um relatório está sendo impresso e uma compilação está sendo executada. Para o usuário tudo se parece como se os três programas estivessem rodando simultaneamente.

De acordo com Tanenbaum (1995), pode-se distinguir dois tipos de sistemas operacionais para sistemas com vários processadores: os sistemas operacionais fracamente acoplados e os fortemente acoplados. Sistemas fracamente acoplados são aqueles onde temos as estações independentes ligadas por rede local. Cada estação tem o seu sistema operacional. Os nodos têm baixo grau de interação e interagem quando necessário. Se a rede local cai, aplicações individuais em cada estação podem continuar. Sistema fortemente acoplado é aquele onde o software integra, fortemente cada nodo da rede. Se a interligação dos processadores for interrompida, a aplicação também será descontinuada. Segundo os conceitos já apresentados tem-se:

- **Sistemas Operacionais de Rede** – Software fracamente acoplado em hardware fracamente acoplado.
- **Sistemas Operacionais Distribuídos** – Software fortemente acoplado em hardware fracamente acoplado.
- **Sistemas Operacionais de Multiprocessadores** – Software fortemente acoplado em hardware fortemente acoplado.

Sistemas Operacionais (SO) de Rede permitem compartilhar diferentes recursos de uma determinada estação entre os demais membros da rede. Um dos recursos mais importantes para ser compartilhado é o disco, permitindo que múltiplas estações possam acessar uma única base de dados. Diversos produtos comerciais estão disponíveis, entre eles Rede Novell da Novell, Inc, WINDOWS NT da Microsoft e NFS (*Network File System*) da SUN Microsystems. O NFS se tornou um padrão de mercado e hoje todos os UNIX têm uma implementação de NFS. Permite que uma determinada área de disco,

um diretório ou conjunto de diretórios, seja exportado para outras máquinas da rede. No servidor existe um arquivo de configuração onde se dá permissão de acesso explicitando qual área e que modo de acesso cada máquina cliente vai estar autorizada nesta interligação. Na máquina cliente se monta um determinado diretório da máquina servidora sob o sistema de arquivos local e a área compartilhada para ser vista como se fosse parte da estrutura de arquivos do sistema local. Qualquer máquina com S.O. Unix pode ser cliente e/ou servidora de um sistema NFS e a Figura 9 (TANENBAUM, 1995), mostra a estrutura em camadas dos componentes envolvidos em uma ligação Cliente / Servidor.

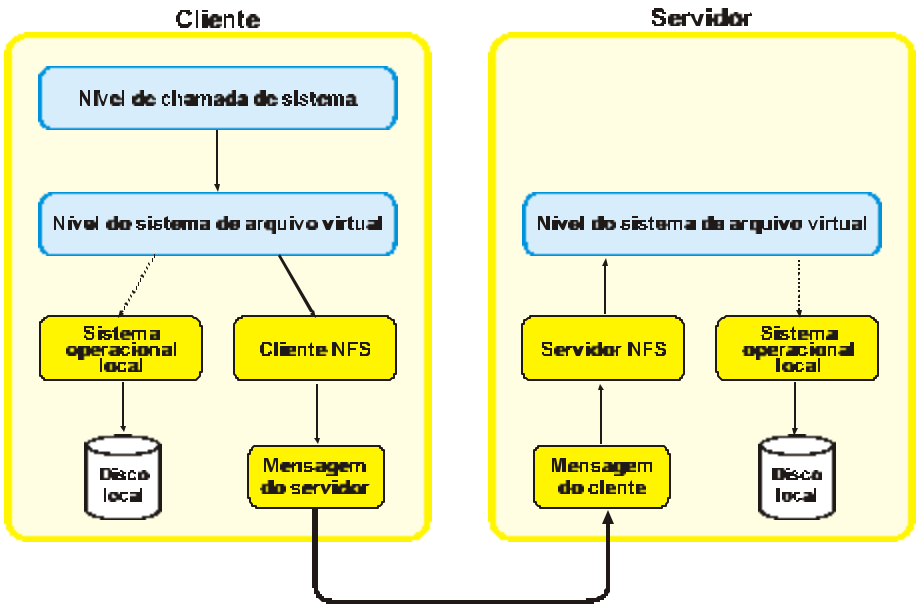

Figura 9 Estrutura em Camadas do NFS

Sistemas verdadeiramente distribuído, vão além do simples uso do NFS. "Um sistema distribuído é aquele que roda em um conjunto de máquinas sem memória compartilhada, máquinas estas que mesmo assim aparecem como um único computador para seus usuários" (TANENBAUM, 1995). Deve haver um sistema de comunicação entre processo, único e global que permita que qualquer processo fale com qualquer outro. Também deve haver um esquema de proteção global. A gerência dos processos também precisa ser a mesma em todo o sistema. A forma como um processo é criado, destruído, inicializado e finalizado, não pode variar de uma máquina para outra.

Sistemas Multiprocessadores de tempo compartilhado, combinação de software fortemente acoplado com hardware fortemente acoplado, são voltados para

processamentos específicos. A característica fundamental desta classe de sistema é a existência de uma única fila de processos prontos: uma lista de todos os processos do sistema que não estão bloqueados, e que só não estão rodando por falta de processador disponível. O sistema da Figura 10 (TANENBAUM, 1995) mostra uma situação com três processadores e cinco processos prontos para rodar.

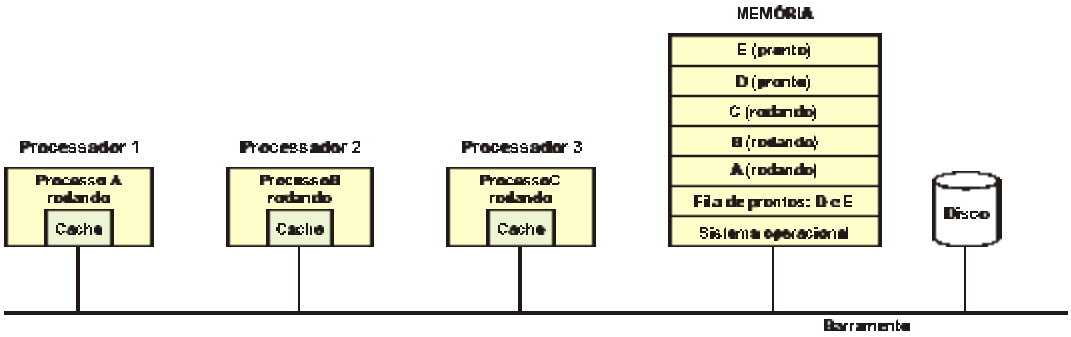

Figura 10 Um Multiprocessador com uma Única Fila de Processos Prontos

Segundo Stallings (1998), um sistema operacional de redes não é atualmente um sistema operacional, mas sim um conjunto distribuído de softwares de sistema para suportar o uso de servidores em uma rede. A máquina servidora provê serviços para a rede como gerenciamento de impressora e compartilhamento de arquivos. Cada computador da rede tem seu próprio sistema operacional. O sistema operacional de redes é simplesmente um complemento ao sistema operacional local que permite a interação entre as máquinas de aplicação com a máquina servidora. Já um sistema operacional distribuído é um sistema compartilhado por uma rede de computadores. Aparece para o usuário final como se fosse um único sistema operacional, mas fornece acesso transparente a recursos de diversas máquinas. Um sistema operacional distribuído pode aproveitar uma arquitetura de comunicação já existente, no entanto mais comumente um conjunto de instruções básicas para funções de comunicações é incorporado ao sistema operacional de forma a prover mais eficiência.

A implementação do sistema uniprocessador virtual só é possível se tivermos memória compartilhada para serem acessadas pelos multiprocessadores. Esta solução para redes de computadores fica muito difícil, senão impossível, devido à baixa velocidade e alta latência no processo de intercomunicação entre os processadores.

2.4.2 Ambientes de Programação Paralela e Distribuída

Do ponto de vista da aplicação, a forma como os programas podem ser desenvolvidos bem como o uso dos recursos existentes, estão relacionados com a arquitetura de processamento paralelo ou processamento distribuído. Segundo Radajewski (1998), tem-se basicamente duas formas de comunicação entre os processos durante a sua fase de execução:

1. Usando *threads* (linhas de execução, parte do código que roda independente), recurso do sistema operacional, em máquinas de memória compartilhada, onde as comunicações são feitas através da memória diretamente, em máquinas do tipo SMP;
2. Via mensagens, em máquinas onde o acesso à memória é feito através de mensagens que são trocadas entre os processos de cada máquina, exemplo Cluster ou interligadas em rede.

Tanto mensagens quanto *threads* podem ser implementadas em SMP, NUMA-SMP e cluster embora eficiência e portabilidade em cada caso, tenham as seguintes considerações:

- **Mensagens** - historicamente a tecnologia de troca de mensagens refletia o projeto dos primeiros computadores paralelos com memória local. Mensagens requerem que os dados sejam copiados enquanto *threads* usam os dados no mesmo local. A latência e a velocidade com que as mensagens podem ser copiadas são os fatores limites neste modelo de troca de mensagens. Uma mensagem é bastante simples: alguns dados e um processador de destino. Alguns mecanismos comuns são interfaces de programas de aplicação tais como PVM (*Parallel Virtual Machine*) ou MPI (*Message Passing Interface*). O mecanismo de troca de mensagens pode ser eficientemente implementado usando *threads* e terá bom resultado tanto em máquinas SMP como entre clusters e máquinas em rede.
- ***Threads*** - as *threads* em sistema operacional foram desenvolvidas devido ao projeto de compartilhamento de memória o que permitiu uma comunicação em memória muito rápida e sincronização entre as partes de programas concorrentes. As *threads* funcionam bem em sistemas SMP por

que a comunicação é feita através da memória compartilhada. O problema com *threads* é a dificuldade de estender o uso além de máquinas SMP pois sendo os dados compartilhados entre CPUs o overhead para atualizar o cache de memória, coerência de cache, seria muito grande. Para estender *threads* eficientemente além dos limites do SMP, seria necessário usar a tecnologia NUMA o que é muito caro.

A programação paralela ou programação distribuída terá realmente vantagem com esta arquitetura de sistemas paralelos e sistemas distribuídos se a aplicação for adequada para este ambiente. Inicialmente considera-se a diferença entre concorrência e paralelismo. Partes concorrentes são partes de um programa que podem ser executados independentemente. O paralelismo é quando temos partes concorrentes deste programa que podem ser executadas ao mesmo tempo em elementos de processamento separados. A distinção é muito importante pois concorrência é uma propriedade do programa e um eficiente paralelismo é uma propriedade do equipamento. A tarefa do programador é identificar que parte concorrente pode ser executada em paralelo e qual parte não pode.

2.4.3 DCE Ambiente de Computação Distribuída

O Ambiente de Computação Distribuída, (DCE – *Distributed Computing Environment*) é uma tecnologia de software desenvolvida pelo Open Group (OPEN, 1999), que permite o desenvolvimento de aplicações distribuídas usando sistemas heterogêneos. O Open Group é um consórcio de usuários e de fornecedores de computadores que trabalham em conjunto visando à tecnologia de sistemas abertos. A organização integra as melhores tecnologias e fornece para que sejam adotadas pelas industrias. Através do Open Group, diversos segmentos da empresas trabalham de forma cooperativa para a evolução do DCE. Algumas áreas críticas que são atendidas por esta tecnologia:

- Segurança
- Internet / Intranet
- Objetos Distribuídos

A tecnologia DCE foi projetada para trabalhar independente de qual sistema operacional ou qual tecnologia de rede está sendo usada pela aplicação. Assim permite

uma interação entre clientes e servidores em qualquer tipo de ambiente, mesmo heterogêneo, que a organização possa ter.

O Open Group fornece o código fonte do DCE os quais os fornecedores incorporam a seus produtos. Muitos fornecedores já agregam funções básicas do DCE junto com os seus sistemas operacionais. A tecnologia inclui serviços de software que residem acima do sistema operacional, fornecendo interface (*middleware)* para os recursos de baixo nível do sistema operacional e recursos de rede. Estes serviços permitem que a organização possa distribuir o processamento e os dados através de toda a empresa. Atualmente DCE é o único ambiente integrado de serviços, não vinculado a um fornecedor específico, que permitem as organizações desenvolverem, usarem e manterem aplicações distribuídas através de redes heterogêneas.

Os serviços DCE incluem:

- RPC – Chamada de procedimento remoto, o que facilita a comunicação cliente-servidor, e então uma aplicação pode efetivamente acessar recursos distribuídos através da rede.
- Serviço de Segurança – Autentica a identidade dos usuários, autoriza o acesso a recursos em redes distribuídas e provê a contabilização de usuários e servidores.
- Serviço de Diretórios – Provê um modelo único de nome através do ambiente distribuído.
- Serviço de Relógio – Sincroniza os relógios dos sistemas através da rede.
- Serviços de *Threads* – Provê a capacidade de execução de múltiplas *threads*.
- Serviço de Arquivos Distribuídos – Provê acesso a arquivos através da rede.

Com relação à segurança, DCE é um dos mais seguros ambientes de computação distribuída. Incorpora a tecnologia Kerberos, uma forma, altamente confiável, bem gerenciada e bem entendida de proteger computação em rede.

Interagindo com a tecnologia hoje existente para internet/intranet a tecnologia para servidores seguros WEB prevê uma integração com os serviços de nome e serviços de segurança do DCE. Assim uma integração do DCE com Secure-http

complementando as facilidades de segurança já existentes com autenticação de usuários para acesso a serviços WEB de informações restritas.

Com relação à tecnologia de objetos distribuídos, DCE lidera a interoperabilidade de diferentes estratégias para objetos distribuídos. DCE permite o uso de objetos distribuídos sem desconsiderar propostas como CORBA. De fato DCE tem os pré-requisitos das especificações CORBA como um ambiente específico Inter Orb Protocol que já está disponível, testado e em uso atualmente. DCE é um componente chave para muitos vendedores implementarem tecnologias de orientação a objetos. Como exemplo a IBM tem acordo para suportar os serviços de segurança, nome e relógio do DCE em seu SOM/DSOM. Hewlett-Packard esta fornecendo DCE++, uma ferramenta orientada a objeto que provê um compilador C++IDL e biblioteca de classes para DCE. A Microsoft usa DCE como base para a comunicação com ActiveX. Digital usa a segurança do DCE em seu ObjectBroker, seu CORBA. Muitos fornecedores têm expressado a sua intenção em usar DCE RPC como um protocolo específico dentro da especificação CORBA para prover interoperabilidade.

2.4.4 Programação Distribuída Orientada a Objeto

A programação orientada a objeto é considerada atualmente um dos melhores modelos de programação para tratar com sistemas complexos enquanto provê facilidade de manutenção, novas implementações e reusabilidade. Um modelo contendo estes atributos é particularmente interessante para sistemas distribuídos, já que estes tendem a se tornar muito complexos. A base da programação orientada a objeto é o conceito de objeto, que é uma entidade que encapsula um estado e permite o acesso através de uma interface bem definida.

Programação distribuída orientada a objeto generaliza a programação orientada a objeto para sistemas distribuídos, estendendo o modelo de objeto com interação remota cliente/servidor entre objetos não locais. No modelo cliente/servidor, o servidor provê acesso aos clientes para serviços específicos através de mecanismos de intercomunicação de processos, IPC, tais como passagem de mensagens ou chamadas de procedimentos remotos. Na programação distribuída orientada a objeto um servidor de objetos encapsula um estado interno e prove o serviço para uma coleção de objetos clientes remotos. Assim como na pura programação orientada a objeto, este serviço é abstraído através da interface, ou seja, uma coleção de métodos que podem ser usadas

para acessar o serviço. Clientes que estão residindo em diferentes máquinas podem invocar estes métodos através da invocação de métodos remotos.

Nos últimos anos tem surgido diversos ambientes de desenvolvimento baseados no paradigma da programação distribuída orientada a objeto. Estes ambientes, entre eles os mais notáveis exemplos são CORBA (OMG, 1998), *JAVA Remote Method Invocation* (RMI) (SUN, 1998), e DCOM (*Distributed Component Objet Model*) (GALLI,1998).

A seguir discutem-se as principais características destes sistemas.

2.4.4.1 CORBA - Common Object Request Broker Architecture

Existe um consórcio internacional de indústrias que promove a teoria e a prática de programação distribuída orientada a objetos que é chamado de OMG – *Object Management Group* (OMG, 1998). Seu objetivo é definir uma arquitetura padrão e comum para que através de diferentes plataformas de hardware e sistema operacionais, possa ser feita a intercomunicação entre objetos. O consórcio OMG foi fundado em 1989 por oito companhias e atualmente conta com cerca de 800 membros, com a participação dos maiores fabricantes de computadores. Esta organização não desenvolve projetos, trabalha com as tecnologias existentes oferecidas pelas empresas membros do consorcio.

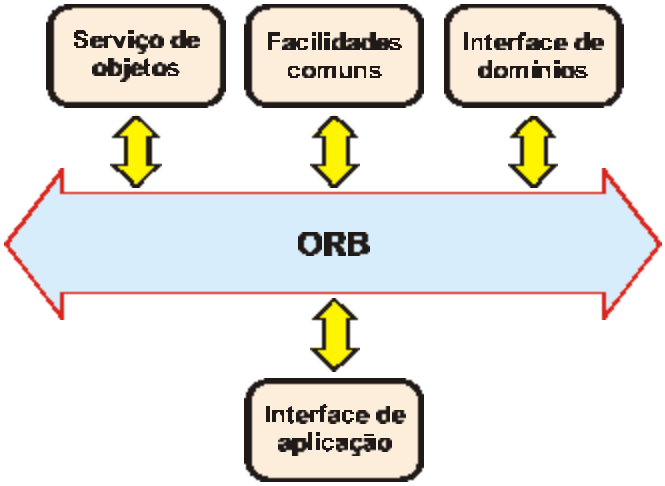

Figura 11 Modelo de Referencia OMG OMA

Sua forma de trabalho é divulgar RFPs – *Request for Proposals* em todos os aspectos da tecnologia de objetos solicitando especificações de novos componentes. Membros podem então propor uma especificação que é acompanhada pela implementação provendo o detalhamento dos conceitos inerentes a esta especificação. Um processo de revisão e votação é conduzido e assim que a especificação é aceita, qualquer fabricante pode implementar esta alternativa em seu produto para o mercado.

O consorcio OMG tem desenvolvido um modelo conceitual, conhecido como *core object model,* e uma arquitetura de referencia, chamada OMA – *Object Management Architecture* sobre a qual as aplicações podem ser construídas. A OMG OMA se propõem a definir em alto nível de abstração, as muitas facilidades necessárias para a computação distribuída orienta a objetos. Assim é proposto um modelo de referencia OMA com cinco partes principais, representados na Figura 11 (OMG,1998), como segue:

- ORB – *Object Request Broker*, (também comercialmente conhecido como CORBA) que é um barramento comum de comunicação para os objetos. Permite que objetos clientes de forma transparente e confiável, requisitem operações em objetos remotos e recebam respostas em um ambiente distribuído. A integração de objetos distribuídos está disponível através de plataformas, independente de sistemas operacionais e serviços de transporte da rede.
- Serviço de Objetos – são componentes de uso geral fundamentais para o desenvolvimento de aplicações CORBA. Um serviço de objetos é basicamente um conjunto de objetos CORBA que podem ser requisitados através do ORB. Serviços não estão relacionados com nenhuma aplicação especifica mas são blocos de construção básica, usualmente provida no ambiente CORBA, suportando funcionalidades básicas úteis para muitas aplicações. Diversos serviços, tem sido projetados e adotados como standard pela OMG (OMG, 1998), incluindo notificação de eventos, interfaces de processamento de transação, segurança, controle de autenticação, etc.
- Interfaces de Domínio – representam áreas verticais que provêem funcionalidades de interesse direto para usuários finais em específicos domínios de aplicação tais como financeiro ou saúde.

- Facilidades Comuns – provêem ao usuário final funções úteis em diversas aplicações que podem ser configuradas para uso específico em aplicações particulares. São facilidades relacionadas com o usuário final, tais como serviço de impressão, gerenciamentode documentos, e correio eletrônico.
- Interface de Aplicação – são específicas para aplicações de usuário final. Representam aplicações baseadas em componentes, executando tarefas especificas para o usuário. Uma aplicação é tipicamente composta de um grande número de objetos, alguns dos quais são específicos da aplicação e outros parte dos serviços de objetos, facilidades comuns ou interface de domínios.

Todos os serviços OMG devem ser especificados usando OMG – IDL (*Interface Definition Language*). Uma linguagem puramente declarativa e não provê detalhes da implementação. É uma linguagem de programação neutra, independente de rede e assim usada como uma forma de descrever os tipos de dados. A sintaxe do OMG IDL é derivada do C++, removendo os construtores de uma linguagem de implementação e adicionando um número de novas palavras chaves necessárias para a especificação de sistemas distribuídos.

Uma definição de CORBA (COULOURIS, 2001), "é um *middleware* que permite aos programas de aplicação se comunicarem entre si, independente de suas linguagens de programação, sua plataforma de hardware e de software e a rede sobre a qual eles se comunicam". As aplicações são construídas com objetos CORBA os quais implementam interfaces definidas na linguagem de definição de interfaces (IDL). Clientes acessam os métodos nos IDL interfaces dos objetos CORBA por meio de invocação método remota (RMI). O componente de *middleware* que suporta RMI é chamado de distribuidor de requisição de objetos (ORB – *Object Request Broker*). A especificação do CORBA foi patrocinada por membros do grupo OMG – *Object Management Group*. Muitos diferentes ORBs foram implementados segundo a especificação, suportando uma variedade de linguagens de programação. Os serviços CORBA provêem facilidades genéricas que podem ser usadas em uma grande variedade de aplicações.

2.4.4.2 JAVA Remote Method Invocation

JAVA *Remote Method Invocation* (RMI) é um modelo de objeto distribuído para a linguagem JAVA que mantém a semântica do modelo de objeto JAVA, tornando a distribuição de objetos fácil de implementar e de usar (EMMERICH, 1999). Na terminologia standard JAVA RMI, um objeto remoto é um cujos métodos podem ser chamados de outra máquina virtual JAVA. Um objeto deste tipo é descrito por um ou mais interfaces remotas, que são interfaces JAVA que declaram qual método do objeto remoto pode ser chamado remotamente. RMI refere-se a ação de invocar um método de uma interface remota em um objeto remoto.

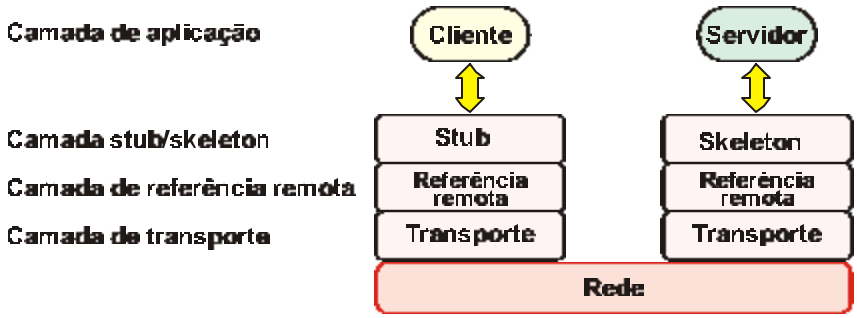

Figura 12 Arquitetura JAVA RMI

A arquitetura JAVA RMI é ilustrada na Figura 12. Usa o mecanismo padrão (derivado de RPC) para comunicação com objetos remotos: *stubs* e *skeletons*. Um *stub* para um objeto remoto atua como um cliente local, representativo para o objeto remoto. Clientes nunca interagem diretamente com objetos remotos, mas somente com objetos *stub* os quais são responsáveis por fazer a chamada em seu remoto objeto destino. Um *skeleton* para um objeto remoto é uma entidade no lado do servidor que contém o método que dispara a chamada para a implementação do atual objeto remoto. *Stubs* e *skeleton* são gerados pelo compilador RMIC fornecido com a distribuição padrão do JAVA RMI.

Um *stub* para um objeto remoto implementa o mesmo conjunto de interfaces remotas implementado pelo objeto remoto. Quando um método do *stub* é chamado, ele faz o seguinte:

- Inicia a conexão com a máquina virtual JAVA remota que contém o objeto remoto.
- Escreve e transmite os parâmetros para a máquina virtual JAVA.
- Espera pelo resultado destas chamadas.
- Lê o valor de retorno ou exceção retornada.
- Retorna o valor a quem chamou.

Na máquina virtual JAVA, cada objeto remoto pode ter um *skeleton* correspondente.(na versão JDK 1.2 *skeletons* não são obrigatórios) O *skeleton* é responsável por disparar a chamada para a atual implementação do objeto remoto. Quando o *skeleton* recebe uma chamada faz o seguinte:

- Lê os parâmetros para o método remoto
- Chama o método na implementação do atual objeto remoto.
- Transmite o resultado para quem chamou.

A implementação da camada de seção no JAVA RMI suporta a ativação de objetos sob demanda através da interface de ativação. Assim evita que no JAVA RMI objetos servidores estejam em memória todo o tempo permitindo que sejam carregados dinamicamente, quando necessário. A ativação é completamente transparente e está representada na Figura 13 (EMMERICH, 1999).

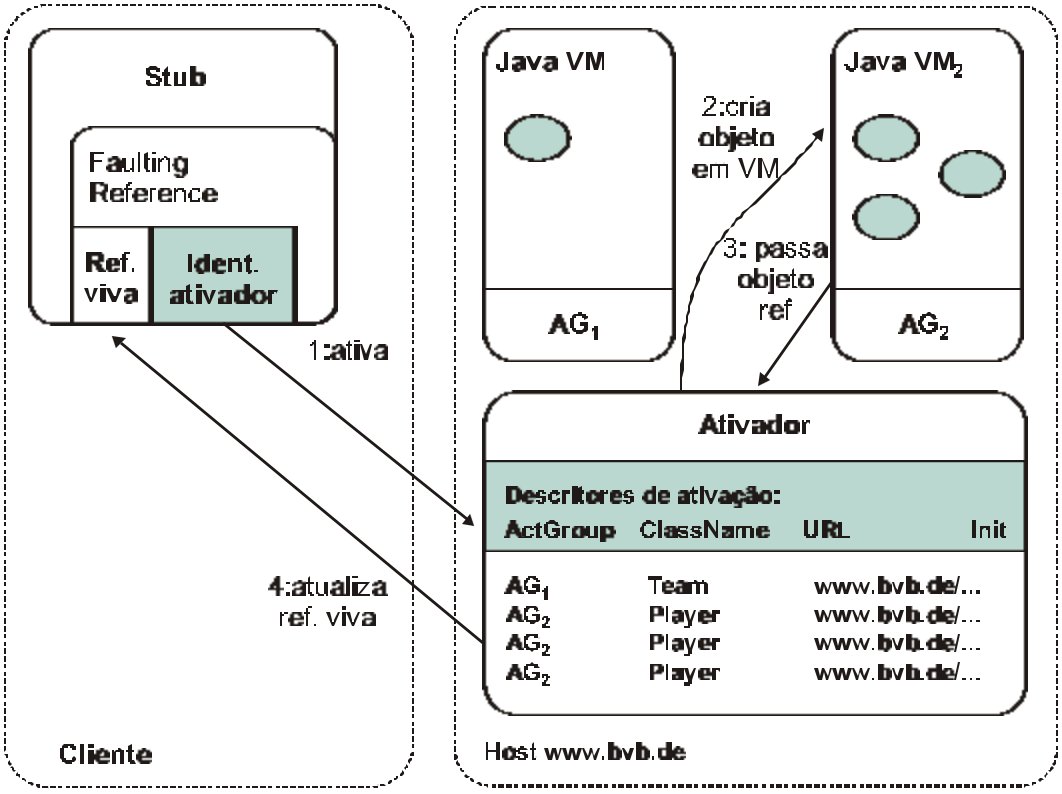

Figura 13 Ativação de Objetos Remotos em JAVA RMI

2.4.4.3 DCOM - Distributed Component Object Model

O modelo de objetos componentes distribuídos (*Distributed Component Objet Model* – DCOM) é o padrão adotado pela Microsoft para distribuição de objetos (GALLI, 1998). Estabelece um modelo de programação, padrão binário, bem como um padrão de interoperabilidade para a computação usando objetos distribuídos. DCOM está disponível desde o WINDOWS NT 4.0 e tinha versões que poderiam ser obtidas na internet para o WINDOWS 95. Está incluído no Microsoft Internet Explorer 4.0, no WINDOWS 98 e disponível para diversas plataformas UNIX fornecido por uma empresa chamada Software AG. Embora iniciado pela Microsoft, COM e DCOM nã o é mais ambiente proprietário da Microsoft. Atualmente o consorcio independente ActiveX é responsável pelo gerenciamento deste padrão.

O modelo básico COM inclui a habilidade de permitir que elementos lógicos sejam considerados independentes. Também permite componentes binários flexíveis para adaptação a diferentes configurações e máquinas. COM é mais largamente

conhecido como a tecnologia básica do ActiveX. Qualquer ferramenta de software que suporta componentes COM, automaticamente suporta DCOM. Existe m quatro serviços no ambiente do servidor ActiveX que são particularmente úteis para DCOM: transações, incluindo a capacidade de recuperação e volta ao estado original; enfileiramento com filas confiáveis de armazenamento e envio que permitem operações em redes sujeitas a intermitentes falta de acesso; uso de script no servidor para permitir fácil integração com aplicações de internet baseadas em HTML; e acesso a sistemas de produção legados. DCOM permite que a tecnologia COM possa se comunicar diretament e através de redes. Alguns pontos importantes do DCOM são:

- Independência de transporte – DCOM permite que os componentes se comuniquem, seja com ou sem orientação a conexão. Suporta TCP/IP, UDP/IP, IPX/SPX, Apple Talk e HTTP.
- Tecnologia aberta – DCOM é uma tecnologia aberta e está disponível para UNIX, Apple, WINDOWS e alguns ambientes mais antigos. Contudo é mais comum no ambiente WINDOWS.
- Integrado com servidor e *browser* WEB – Como DCOM inclui ActiveX e os componentes ActiveX podem ser incluídos em aplicações baseadas em *browser* DCOM permite distribuir aplicações internet que suportam tecnologias de *browser*.
- Segurança – DCOM pode integrar segurança na internet baseada em certificados.
- Extensão do mecanismo RPC do Ambiente de computação distribuída (DCE) – DCOM usa e estende este mecanismo de RPC do DCE.

A arquitetura básica DCOM permite que uma aplicação possa ser desenvolvida de maneira que automaticamente permita uma distribuição futura e com escalabilidade. Assim se uma demanda maior de uma aplicação ac ontece podemos aumentar a capacidade do servidor para atender esta necessidade. Entretanto se instalarmos um novo servidor e a aplicação estiver no padrão DCOM poderemos distribuir a execução sendo que parte da aplicação poderá rodar no servidor antigo e parte no novo servidor. Para se comunicar com um componente que não seja local, DCOM emprega um mecanismo de comunicação inter processos que é completamente transparente para a aplicação. Especificamente DCOM substitui o mecanismo de comunicação local por um

protocolo de comunicação em rede. Desta forma DCOM permite independência e transparência de local para a aplicação.

2.4.4.3.1 Monikers

Os nomes de instância em DCOM são referenciados como *monikers*. São por si só objetos e permitem extrema flexibilidade. Identificadores que vão desde um nome de banco de dados, um servidor, até um endereço URL ou uma pagina HTML podem ser caracterizados como uma instância de um objeto. *Monikers* contém as informações necessárias e lógicas para localizar uma corrente instância que está sendo executada do objeto que ele está se referindo. As instâncias que estão sendo executadas, são listadas na tabela ROT – *Running Object Table*. Esta tabela é usada pelo *Monikers* para encontrar instâncias de objetos que estão em execução, rapidamente.

Monikers armazenam informações de referencia sobre o estado de um objeto. Estas informações estão tipicamente armazenadas em um banco de dados. Um objeto pode indicar para o moniker qual é a atividade quando o estado do objeto for armazenado. Assim uma operação de acesso a dados pode prevenir um possível gargalo se o objeto rodar no mesmo servidor onde estão os dados. Se múltiplos clientes necessitam acessar o objeto, o objeto precisa executar em um lugar acessível a todos os clientes.

2.4.4.3.2 Chamadas a Métodos Remotos

DCOM executa uma chamada a método remoto quando um cliente quer chamar um objeto em outro espaço de endereçamento. No DCOM a linguagem de descrição de interface (IDL) foi construída em cima do padrão DCE RPC. Como é usual em RPC todas as informações parâmetros são disponibilizadas após serem concatenadas em um único espaço de memória, sendo passado o endereço desta área. O cliente faz o processo inverso e a partir do endereço, recria os dados que o processo vai receber. O retorno desta chamada volta com parâmetros segundo o mesmo processo anterior. O lado do cliente é conhecido como *proxy* e o lado do servidor e conhecido como *stub.* Um tipo adiciona de dados não incluído em DCE RPC é o ponteiro de interface. Estes tipos de ponteiros podem aparecer como resultado da CoCreateInstance ou como parâmetro para

a chamada de um método. Para tratar este novo tipo de dado a criação de um par *proxy/stub* é feita com a capacidade de tratar este novo tipo de dados em todos os métodos da interface.

2.4.4.3.3 Coleta de Lixo

O mecanismo primário para controle do tempo de vida de um determinado objeto é um contador de referencia. Este contador incrementa quando se usa AddRef e decrementa com o Release. Como nem todos os clientes terminam normalmente, um processo de Ping é empregado. Cada objeto exportado tem um tempo de PingPeriod e um contador de numPingsToTimeOut. A combinação destes valores determina o tempo total conhecido como período de ping. Se este tempo passa sem que ocorra um *ping* para o referido objeto, a referencia remota é considerada expirada. O contador de referencia é decrementado como se o processo tivesse terminado normalmente. Quando um objeto não tem referencia ativa, um processo de coletor de lixo se encarrega de liberar a memória então ocupada por este objeto.

2.4.4.3.4 Modelo de *Thread* Suportado em DCOM

COM e DCOM operam com a capacidade de *multithread* nativa do sistema operacional. Assim a comunicação inter processo no DCOM requer DCE RPC, que por sua vez requer que mensagens RPC sejam processadas em uma *thread* arbitrária. DCOM provê sincronização automática do método de chamada para uma *thread* simples. DCOM usa o modelo apartamento com relação ao tratamento de *threads*. Com relação aos parâmetros podemos pensar em cada *thread* como um processo separado. Acesso na mesma *thread* é acesso direto enquanto de diferentes *thread* é indireto, possivelmente através do proxy/stub. Um objeto pode suportar qualquer dos modelos de *thread* que segue:

- Apartamento *Thread* única, *thread* principal somente – Neste modelo todas as instâncias são criadas na mesma *thread* única associada com este objeto.

- Apartamento *Thread* única – Neste modelo uma instância é amarrada a uma *thread* única contudo diferentes instâncias podem ser criadas em diferentes *threads*.
- Apartamento *Multithread* – Neste modelo instâncias podem ser criadas em múltiplas *threads* e podem ser chamadas em *threads* arbitrarias.

Qualquer objeto implementa como o servidor local controla o tipo de apartamento usado para seu objeto. Quando um objeto está sendo criado, COM necessita saber se o objeto é compatível com o modelo de *thread* do objeto pai. Se o objeto filho não é compatível, COM tenta carregar o objeto em um apartamento diferente que seja compatível.

2.4.4.3.5 Segurança em DCOM

Assim como qualquer sistema que suporta computação distribuída DCOM também tem seus aspectos de segurança:

- **Segurança de acesso** – Assegura que um objeto possa ser chamado somente por objetos que tenham a permissão apropriada.
- **Segurança no lançamento** – Assegura que somente objetos apropriados possam criar novos objetos em um novo processo.
- **Identidade** – O principio de como um objeto identifica a si mesmo.
- **Políticas de conexão** – Se uma mensagem pode ser alterada e se uma mensagem é capas de ser interceptada por outro objeto.

2.4.4.4 Pontos Chaves de CORBA, JAVA e DCOM

Segundo Emmerich (1999), quando avalia-se CORBA, JAVA e DCOM encontramos os seguintes pontos chaves:

- CORBA, DCOM e JAVA/RMI permitem que objetos clientes requisitem a execução de operações de objetos servidores distribuídos. Estas operações são parametrizadas e detalhados mecanismos de passagem de parâmetros são especificados.

- CORBA, DCOM e JAVA/RMI todos usam uma forma de referência para identificar objetos servidores de forma transparente quanto à localização. Objetos CORBA são identificados por referencia ao objeto, objetos COM são identificados por ponteiros de interface e objetos JAVA/RMI são identificados por referencias de falta.
- JAVA/RMI é diferente de CORBA e DCOM pelo fato que integra objetos não remotos ao modelo objeto. Isto é obtido modificando o mecanismo padrão de passagem de parâmetros no modelo e passando objetos não remotos por valor.
- O modelo objeto do CORBA, DCOM e JAVA/RMI, todos separam a noção de interface e implementação. A separação é mais específica em CORBA e DCOM, onde existem linguagens separadas para definir interfaces e onde a implementação pode ser escrita em diferentes linguagens de programação.
- Os modelos de objetos dos três ambientes suportam herança. Todos especificam uma classe primária que é usada para derivar propriedades comuns a qualquer objeto referenciado. Esta classe principal é Object para CORBA, IUnknown im DCOM e Remote in JAVA/RMI.
- Nos três enfoques, atributos são tratados como operações. Pode ser implícito como em CORBA ou explicitamente incorporado pelo projetista usando DCOM ou JAVA/RMI.
- Todos os modelos de objeto têm suporte para tratar com falhas durante a requisição de objetos.
- Todos os modelos de objeto são estaticamente de um tipo (tipo de variável) e suportam uma forma restrita de polimorfismo. Em todos os modelos é possível associar um objeto a uma variável de diferente tipo desde que o tipo da variável estática seja um supertipo do tipo do objeto dinâmico.
- Em todos os três sistemas de *middleware* tem-se geradores para os clientes e *stubs* no servidor para implementar a camada de apresentação. CORBA usa *stubs* no cliente e implementa *skeletons.* DCOM usa interface *proxie* e interface *stub.* Em JAVA/RMI eles são referenciados como *stubs* e *skeletons*.

- CORBA, COM e JAVA/RMI todos suportam a ativação sob demanda de objetos servidores. CORBA tem esta facilidade através de adaptador de objeto; em DCOM é implementado pelo gerenciador de controle de serviço; em JAVA/RMI o ativador implementa esta ativação.

2.5 Comunicação nos Sistemas Distribuídos

A comunicação em sistemas distribuídos se torna mais complexa, pois a troca de informações entre dois processos localizados em máquinas distintas depende primeiramente da interligação destas máquinas. Quando em um único equipamento dois processos podem se comunicar através da memória compartilhada que é comum e acessível aos dois processos simultaneamente. Segundo Stallings (1998), a interligação dos equipamentos hoje pode ter vários enfoques desde o tratamento de um computador pessoal como um simples terminal até um alto grau de integração entre aplicações em computadores pessoais e servidores de banco de dados.

Uma das arquiteturas que se tornou padrão de mercado é o TCP/IP (*Transmission Control Protocol/Internet Protocol*), amplamente usado nos dias atuais e que será visto com mais detalhes na seqüência.

2.5.1 Arquitetura do Protocolo TCP/IP

Segundo Stallings (1998), TCP/IP é o resultado de um protocolo pesquisado e desenvolvido na rede experimental ARPANET, fundada pela DARPA (*Defense Advanced Research Projects Agency*) e genericamente conhecido como conjunto de protocolos TCP/IP. Este conjunto de protocolos foi transformado em padrões internet pelo IAB (*Internet Architecture Board*). Não há um modelo oficial de protocolo TCP/IP como no caso do modelo OSI. Contudo, baseados nos protocolos padrões que foram desenvolvidos nos podemos organizar as tarefas de comunicação para o TCP/IP em cinco níveis independentes:

- **Camada de Aplicação** – contém a lógica necessária para suportar as várias aplicações de usuários. Para cada tipo diferente de aplicação, como exemplo a transferência de arquivos, um módulo separado é necessário para esta aplicação peculiar.

- **Camada de transporte** – Independente da natureza da aplicação, os dados que estão sendo trocados entre as aplicações precisam ser confiáveis. O controle da ordenação dos pacotes para garantir que sejam recebidos na mesma ordem que são transmitidos é uma tarefa que independe da natureza da aplicação. O protocolo de controle de transmissão (TCP) é o mais comum nesta camada.
- **Camada de internet** – Quando temos a interligação de diferentes redes é necessário o uso de rotinas que tratam o roteamento para esta conexão. Os procedimentos para tratar estas funções estão na camada internet e o protocolo IP é usado nesta camada para prover as funções de roteamento através de múltiplas redes.
- **Camada de acesso à rede** – Esta relacionada com a troca de dados entre um sistema final e uma rede ao qual está ligada.O computador que envia precisa prover o endereço da rede de destino. O computador que envia pode requisitar certos serviços, tal como prioridade, que precisam ser atendidos pela rede. O software específico usado nesta camada depende do tipo de rede a ser usado; diferentes padrões foram desenvolvidos para o chaveamento de circuitos, chaveamento de pacotes (X.25), rede local (Ethernet) e outros.
- **Camada física** – atende a interface de ligação física entre a estação de transmissão de dados (computador, estação de trabalho) e o meio de transmissão ou rede. Esta camada está relacionada com características dos meios de transmissão, a natureza dos sinais, a taxa de transferência e outros itens relacionados.

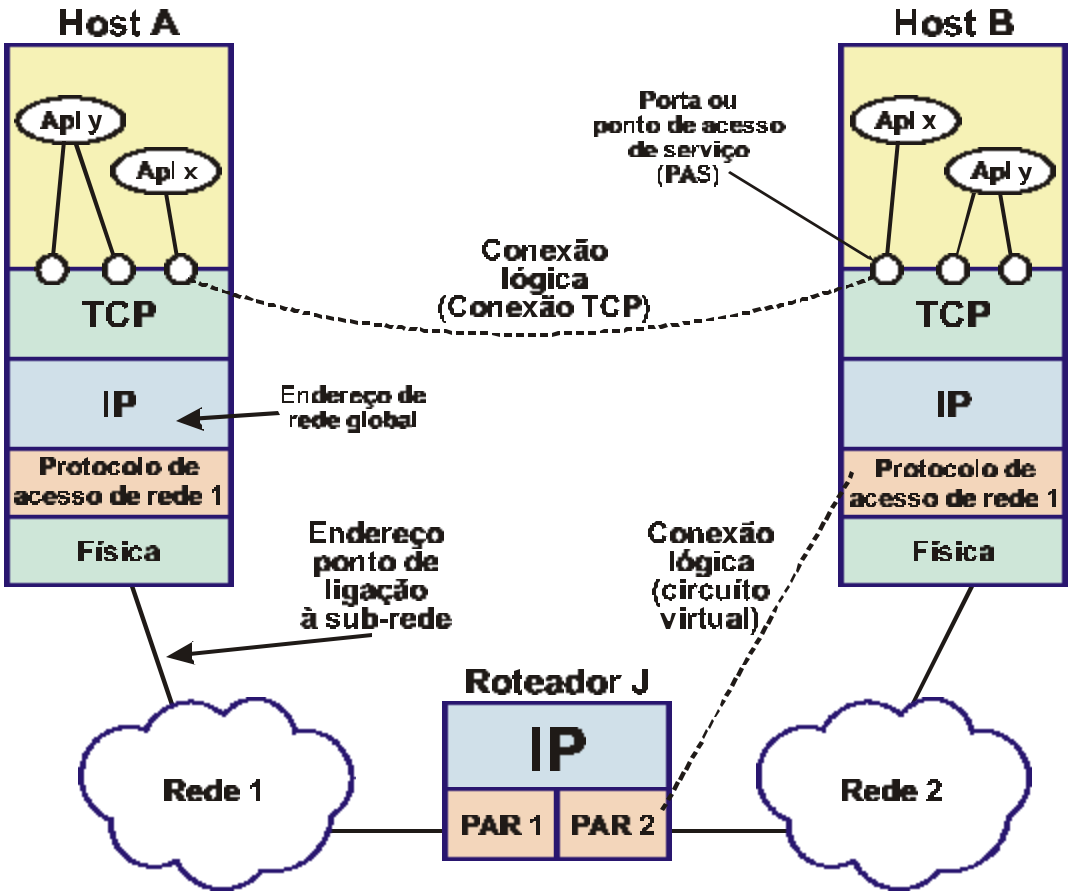

Figura 14 Conceito TCP/IP

Em termos de operação, pode-se ver na Figura 14 como estes protocolos são configurados para comunicação. Para ficar claro que as facilidades de comunicação abrangem múltiplas redes a rede onde estamos conectados é chamada de subrede. Assim um protocolo de acesso como o Ethernet é usado para conectar o computador a subrede. Este protocolo permite enviar dados para outro computador na mesma subrede ou a um roteador nesta subrede. O protocolo IP está implementado em todos os sistemas finais e nos roteadores. Atua como um retransmissor para mover os pacotes da origem ao destino final passando por roteadores que interligam as subredes. TCP é implementado somente nos sistemas finais; responsável por controlar os blocos de dados de forma a que sejam entregues corretamente para as aplicações apropriadas.

Para o sucesso da comunicação, cada entidade no sistema como um todo, precisa ter um endereço único. Atualmente dois níveis de endereçamento são necessários. Cada computador em uma subrede precisa ter um endereço global único o que permite que o

dado seja enviado ao computador específico. Este endereço é usado pelo IP para roteamento e entrega dos pacotes. Em um computador, cada aplicação precisa ter um endereço único neste computador. Este endereço único relativo a aplicação, também chamado de porta, permite ao protocolo TCP fazer a entrega das informações ao processo específico.

2.5.2 Modelo Cliente Servidor

Segundo Tanenbaum (1995), uma das estruturas mais comuns para os sistemas distribuídos é a do cliente-servidor. A idéia por trás deste modelo é a de estruturar o sistema operacional como um grupo de processos cooperantes, denominados servidores, que oferecem serviços a processos usuários, denominados clientes. As máquinas clientes e máquinas servidoras normalmente rodam o mesmo *microkernel*, onde tanto os clientes quanto os servidores rodam como processos usuários. Uma máquina pode rodar um único processo, vários clientes, vários servidores, ou uma mistura dos dois.

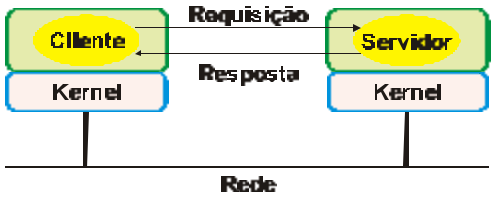

Figura 15 O modelo Cliente Servidor

Para evitar o *overhead* considerável dos protocolos orientado a conexão tal como o OSI ou o TCP/IP, o modelo cliente servidor é baseado em um protocolo muito simples, sem conexão, do tipo solicitação/resposta. A vantagem deste modelo, (Figura 15) é a simplicidade. O cliente envia uma solicitação ao servidor e recebe dele uma resposta. Nenhum tipo de conexão deve ser estabelecido antes do envio da solicitação, nem desfeito após a obtenção da resposta. A própria mensagem de resposta serve como uma confirmação do recebimento da solicitação. Devido a esta estrutura extremamente simples, os serviços de comunicação fornecidos pelo *microkernel* podem por exemplo, ser reduzidos a duas chamadas de sistema, uma para envio de mensagens e outra para a recepção das mesmas.

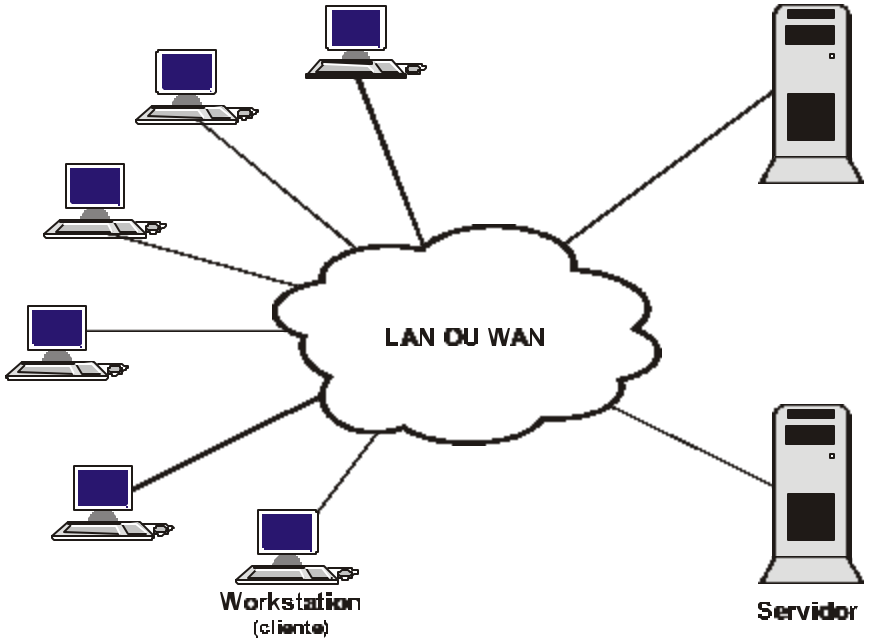

Figura 16 Ambiente Genérico para Cliente Servidor

Segundo Stallings (1998), um exemplo de ambiente genérico para cliente servidor é mostrado na Figura 16. A máquina cliente, geralmente um PC ou estação de trabalho, que tem uma interface muito amigável, geralmente gráfico, para o usuário final. Cada máquina servidora disponibiliza um conjunto de serviços compartilhados para os clientes.

A principal característica da arquitetura cliente/servidor é a alocação de tarefas no nível de aplicação entre clientes e servidores. A Figura 17 ilustra o caso geral. Em ambas, cliente e servidor, o software básico é o sistema operacional rodando na plataforma de hardware. A plataforma e o sistema operacional do cliente pode ser diferente do servidor. Assim o importante é que um servidor e um particular cliente compartilhem os mesmos protocolos de comunicação. É o software de comunicação que permite a interoperabilidade entre o cliente e o servidor. Exemplos destes softwares incluem TCP/IP e OSI. Um exemplo que bem ilustra o processamento cliente/servidor é o caso de aplicações que acessam bancos de dados. Usando linguagem SQL (*Structured Query Language*) um aplicativo cliente pode solicitar dados de tabelas contidas em um servidor de banco de dados. Após processar a requisição o Banco de Dados devolve os resultados para o cliente que ira tratar as informações e mostrar o resultado ao usuário,

geralmente em um ambiente de telas gráficas, dando uma melhor apresentação ao resultado final com os dados obtidos.

Figura 17 Arquitetura Genérica Cliente Servidor

2.5.3 RPC Chamada Remota de Procedimentos

RPC (*Remote Procedure Calls*) é um método comum e largamente aceito para o encapsulamento de mensagens em um sistema distribuído (STALLINGS, 1998). A essência desta técnica é permitir que programas em diferentes máquinas possam interagir usando simples semântica de procedimentos *call/return*, como se os dois programas estivessem na mesma máquina. Isto é a chamada de um procedimento é usada para acessar serviços remotos. A popularidade deste enfoque é devido as seguintes vantagens:

- A chamada de um procedimento é amplamente aceita, usada e de abstração entendida.
- O uso de chamadas remotas de procedimentos permite que interfaces remotas sejam designadas como um conjunto de operações nomeadas com tipos designados. Assim esta interface pode ser claramente documentada e os programas distribuídos podem ser conferidos estaticamente, em tempo de compilação, para validar os erros de tipos (permite conferir os tipos de dados que estão sendo usados nas chamadas, em tempo de compilação)

- Devido à especificação de uma interface padronizada e precisa o código de comunicação para uma aplicação pode ser gerado automaticamente.
- Devido à especificação de uma interface padronizada e precisa o programador pode escrever módulos clientes e módulos servidores que podem ser usados em diferentes computadores e sistemas operacionais com pequenas modificações e pouca recodificação.

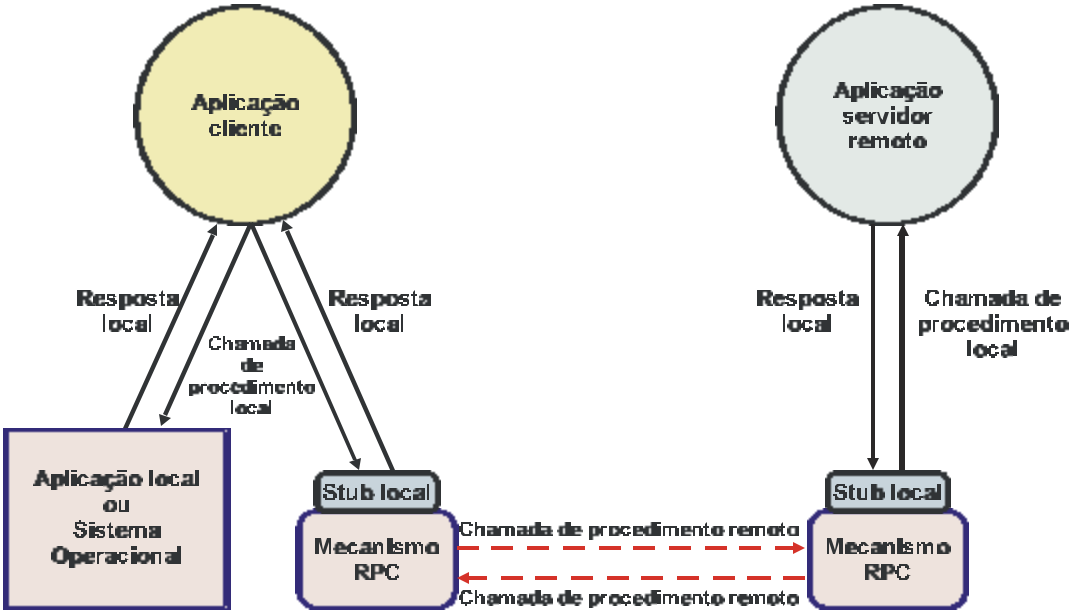

Figura 18 Mecanismo de Chamada Remota a Procedimento

Na Figura 18 tem-se uma visão detalhada do mecanismo de RPC. O programa que chama faz uma chamada comum de procedimentos com parâmetros em sua própria máquina. Por exemplo a chamada CALL P(X,Y) onde P = Nome do procedimento, X = Parâmetro enviado e Y = Parâmetro recebido

Isto pode ou não ser transparente ao usuário que a intenção é chamar um procedimento remoto em outra máquina. Um pequeno procedimento, chamado de STUB, P deverá ser incluído no procedimento que chama ou ser ligado dinamicamente na hora da chamada. Este STUB cria uma mensagem que identifica o procedimento sendo chamado e inclui os parâmetros. Envia a mensagem ao sistema remoto e aguarda a resposta. Quando uma resposta é recebida este STUB P retorna ao programa chamador passando os valores que retornaram.

Na máquina remota, um outro STUB é associado com o procedimento chamado. Quando uma mensagem chega, ela é examinada e uma chamada local CALL P(X,Y) é gerada. Este procedimento remoto é assim chamado localmente e tratado como uma chamada local.

São características básicas que se pode observar no mecanismo RPC:

- Passagem de parâmetros – Geralmente por valor e não por referência pois os dados dos parâmetros são copiados dentro da própria mensagem enviada.
- Representação dos parâmetros – Quando se tem ambientes heterogêneos pode-se ter diferença na representação de números e até de texto (tipos inteiros, tipos float, ASCII/EBCDIC).
- Forma de ligação – A forma de ligação entre o cliente e o servidor pode ser de forma persistente e não persistente. Diz-se persistente quando é estabelecida a comunicação na primeira chamada e esta ligação permanece após a primeira resposta. De outra forma quando a cada mensagem é estabelecida uma comunicação, enviada a mensagem, recebida a resposta e desfeita a comunicação, chamamos de ligação não persistente.
- Sincronismo – O tradicional é a chamada de procedimentos de forma síncrona, ou seja o procedimento que chamou, passa a mensagem e fica aguardando a resposta do procedimento chamado. Para prover flexibilidade, várias facilidades de chamadas de procedimentos assíncronas foram implementadas visando um maior grau de paralelismo enquanto mantêm a simplicidade das chamadas de procedimentos remotos.

Segundo Tanenbaum (1995), o objetivo da chamada remota de procedimentos é manter escondido do usuário todos os procedimentos relativos à comunicação remota. Algumas dificuldades se apresentam quando temos a ocorrência de erros no servidor, no cliente ou na rede gerando situações como segue:

- **O cliente não é capaz de localizar o servidor**: Devido a uma falha de hardware e o servidor efetivamente não estar disponível.
- **Perda de mensagem solicitando serviço**: Usando um contador de tempo, se o tempo expira e a resposta não veio, retransmite a mensagem.
- **Perda de mensagem com resposta:** Uma solução obvia é um temporizador e se depois de determinado tempo o cliente não obtém a

resposta, faz a requisição novamente. O problema é que nem sempre podemos simplesmente pedir para repetir a operação. Ocorrem situações onde a requisição pode ser feita repetidas vezes e teremos sempre o mesmo resultado. Este tipo de operação é chamado **idempotente**. Outra situação é se repetimos a solicitação e os dados são alterados no servidor a cada chamada refeita. Exemplo, adicionar um valor ao saldo de uma conta corrente. Nesta situação temos uma operação **não idempotente**.

- **Quedas do servidor:** Este tipo de problema também está relacionado com a operação idempotente mas não pode ser resolvida com o uso de números seqüenciais nas mensagens. Num servidor a ordem normal dos eventos é receber a solicitação, executar o procedimento e responder. Assim uma parada no servidor pode acontecer após a execução ou antes da execução do procedimento. Existe a técnica de chamada de no mínimo uma vez, e garante que a chamada remota ao procedimento será executada no mínimo uma vez. Outro método é chamado de semântica de no máximo uma vez e garante que a chamada remota foi executada no máximo uma única vez.
- **Queda do cliente:** É a situação quando o cliente manda uma requisição e antes de receber a resposta, sai do ar.

Na chamada remota a procedimentos o protocolo empregado é muito importante. Em princípio qualquer protocolo que permita enviar os bits do cliente para o servidor e vice-versa, poderia ser usado, porém o aspecto de performance é muito importante e está diretamente relacionado com o protocolo adotado. Pode-se usar protocolo orientado a conexão ou protocolo não orientado a conexão, que não necessitam estarem sempre interligados para a troca de mensagens. Quando se usa um protocolo orientado a conexão, antes de enviar a primeira mensagem entre dois procedimentos, é estabelecida uma conexão a qual ficará ativa a partir deste ponto. Assim facilita muito o processo de retransmissão, pois o tratamento dos erros será feito pelo protocolo em um nível abaixo e não pela aplicação que troca mensagens. Em especial em ligações de longa distância é usado um protocolo orientado a comunicação. Entretanto para redes locais o nível de erros é muito baixo e o *overhead* devido ao uso de um protocolo orientado à conexão é muito alto. Neste caso um protocolo sem conexão será empregado na maioria das vezes. Outra consideração é se devemos usar um protocolo padrão, de propósito geral, ou devemos desenvolver um específico para chamada remota. Alguns sistemas distribuídos usam o IP, ou o UDP que é construído em cima do IP (TCP/IP), como protocolo básico.

A decisão de implementação a partir de um protocolo padrão como TCP/IP é devido a facilidades existentes tais como: já estar desenvolvido e testado; padrão em todos sistemas UNIX; disponível em diferentes sistemas operacionais e adequado a redes locais e redes remotas.

Com relação à performance, o IP não foi projetado para ser um protocolo para usuário final. Seu projeto baseia-se no fato de que é possível estabelecer conexões TCP confiáveis entre redes.

Outro aspecto muito importante a ser considerado é o fluxo de controle. Muitos *chips* de interface de rede são capazes de enviar pacotes consecutivamente sem quase nenhum intervalo entre eles, más estes mesmos *chips* normalmente não têm como receber um número ilimitado de pacotes, devido à sua capacidade de armazenamento ser finita.

Com relação a confirmações de mensagens tem-se a situação do servidor eventualmente perder a confirmação de uma mensagem. Na chamada remota de procedimentos o protocolo consiste em uma requisição, uma resposta e uma confirmação. Esta última é necessária para fazer com que o servidor possa desfazer-se das respostas que tenham sido recebidas corretamente pelo receptor. Na prática o servidor pode inicializar um temporizador ao enviar a resposta e livrar-se dela quando este temporizador expirar. Outra forma é interpretar uma nova requisição do cliente, como sendo também uma confirmação da requisição anterior, pois se a anterior não fosse recebida, não estaria fazendo uma nova requisição.

Alguns aspectos sobre gerência de tempo devem ser considerados na chamada remota de procedimentos. Todos os protocolos são desenvolvidos para permitir a troca de mensagens sobre algum tipo de meio de comunicação. Em todos os sistemas, algumas mensagens podem ser ocasionalmente perdidas, devido ao ruído na comunicação ou ao *overflow* no *buffer* de recepção. Em conseqüência, a maioria dos protocolos inicializa um temporizador sempre que uma mensagem é expedida e uma resposta é esperada. Se a resposta não chegar em determinado tempo, o temporizador sinaliza, e a mensagem original é retransmitida.

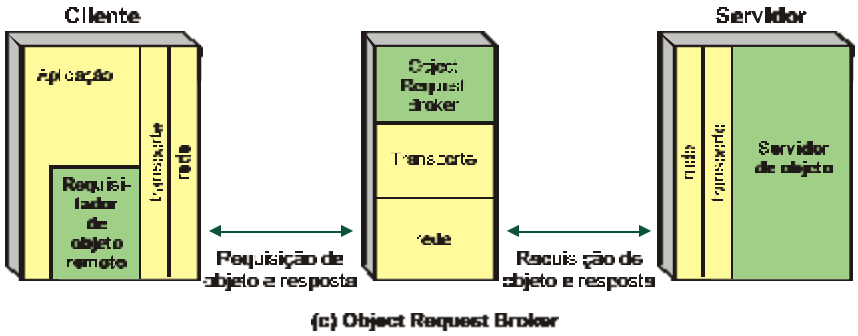

Figura 19 Servidor de Requisição de Objetos

Em um nível superior a chamada de procedimentos remotos, encontramos o mecanismo de comunicação de objetos. Nos mecanismos de comunicação com orientação a objetos podemos ter simples mensagens ou objetos completos sendo trocados entre os processos que se comunicam. Segundo Stallings (1998), um cliente que necessita um serviço envia uma requisição para um provedor de requisição de objetos que atua como um diretório de todos os serviços remotos disponíveis na rede, Figura 19. Uma padronização neste mecanismo de troca de objetos ainda não ocorreu. Encontra-se concorrentes como Microsoft com COM/OLE (*Common Object Model / Object Linking and Embedding*) e outros como IBM. Apple, Sun que adotaram CORBA (*Common Object Request Broker Architecture*).

3 COMUNICAÇÃO EM GRUPO

Destaca-se neste capítulo o embasamento conceitual relativo ao tema principal deste trabalho que é a Comunicação em Grupo. Todas as características deste paradigma usado em sistemas distribuídos se encontram conceituadas neste capitulo.

3.1 Introdução

Em sistemas distribuídos é fundamental a comunicação entre os diversos computadores que compõem este ambiente de sistemas distribuídos. Como já se viu anteriormente a Chamada Remota de Procedimentos – RPC se aplica em especial para a comunicação que envolve dois processos. Em algumas situações é desejável que um processo possa se comunicar com diversos outros processos. Com RPC não se pode ter um único transmissor que envia de uma única vez para múltiplos receptores. Este mecanismo de comunicação que trata múltiplas conexões é chamado de Comunicação em Grupo e segundo Tanenbaum (1995), tem as peculiaridades e características que serão apresentadas neste capítulo. Um grupo é uma coleção de processos que interagem entre si em algum sistema. A propriedade chave que todos os grupos tem é que quando uma mensagem é enviada para o grupo, todos os membros do grupo recebem esta mensagem. Esta forma é chamada Comunicação Um-Para-Muitos, um transmissor para muitos receptores, em contraste com a Comunicação Ponto-a-Ponto, conforme ilustrado na Figura 20.

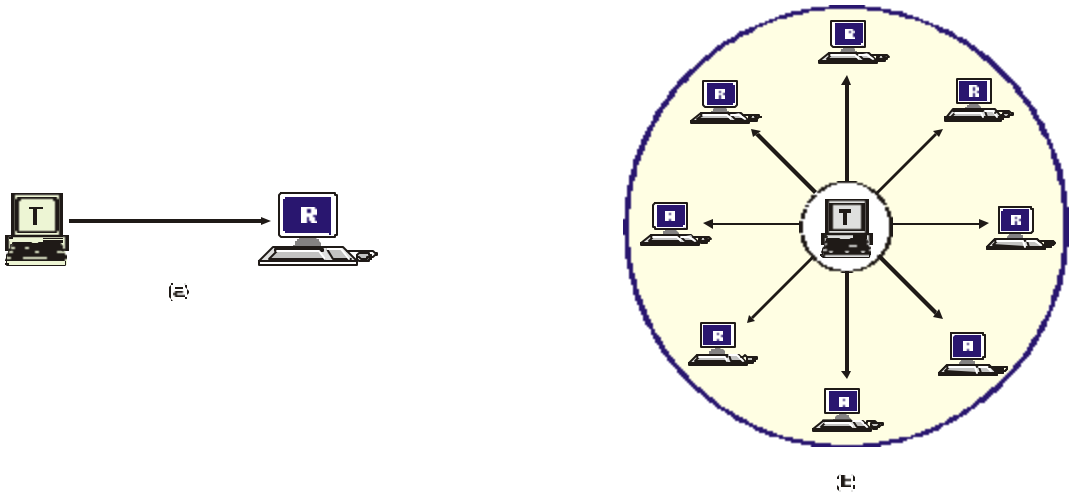

Figura 20 (a) Comunicação Ponto-a-Ponto (b) Comunicação Um-Para-Muitos

Grupos são dinâmicos e assim novos grupos podem ser criados e grupos antigos podem ser destruídos. Um processo pode juntar-se a um grupo ou deixar um grupo. Um processo pode ser membro de diversos grupos ao mesmo tempo. Assim mecanismos são necessários para gerenciar grupos e os processos participantes dos grupos. Quando usamos mecanismos de comunicação em grupo os processos tratam com coleções de outros processos de forma abstrata. Se um processo envia uma mensagem para um grupo de servidores, este processo não precisa se preocupar quantos são os servidores nem onde eles estão localizados. Inclusive já na chamada seguinte o número de servidores deste grupo e suas localizações podem mudar sendo esta alteração transparente para o processo que envia a mensagem para o grupo.

Como será implementada a comunicação em grupo, depende das características do hardware usado na interligação dos equipamentos. Em algumas redes é possível se criar um endereço especial de rede onde múltiplas máquinas podem escutar neste endereço. Quando um pacote é enviado para um destes endereços, automaticamente ele é recebido por todas as máquinas que estão escutando este endereço. Esta técnica é chamada de "*multicasting*". Implementar comunicação de grupos usando "*multicasting*" é simples, basta atribuirmos um endereço de "*multicasting*" para cada grupo. Em redes que não temos o "*multicasting*", ainda podemos ter o "*broadcasting*", onde os pacotes que tem um determinado endereço são escutados por todas as máquinas. O "*broadcasting*" pode ser usado para implementar comunicação em grupo, embora não seja tão eficiente, pois todos na rede vão receber as mensagens e cada processo terá que avaliar se aquela mensagem recebida é para o grupo ao qual ele pertence. Se não for vai descartar a mensagem mas neste caso o processo já gastou tempo e recursos. Mesmo assim ainda tem a vantagem que um único pacote enviado vai atingir a todos os equipamentos da rede. Se a rede não suporta "*multicasting*" nem "*broadcasting*" ainda é possível se implementar um sistema onde o transmissor vai enviar um pacote separado para cada um dos receptores do grupo. Para um grupo com N membros é necessário o envio de N pacotes ao invés de um único pacote como no "*broadcasting*" ou "multicasting". Quando a comunicação é um a um é chamada de "*unicasting*". Além das características normais encontradas nos mecanismos de troca de mensagens, tais como, buferização, blocagem, no tratamento de grupos temos novas características de organização, endereçamento e outras que serão detalhadas na seqüência.

3.2 Organização dos Grupos

Os sistemas que suportam comunicação em grupo podem ser divididos em duas categorias, dependendo que quem está enviando para quem. Alguns sistemas suportam o conceito de grupo fechado, no qual somente membros do grupo podem enviar mensagens para o grupo. Membros de fora do grupo não podem enviar mensagens para o grupo como um todo, embora possam enviar mensagens para membros individuais. Outro conceito, grupo aberto, onde qualquer processo pode enviar mensagem para qualquer grupo. Na Figura 21 (TANENBAUM ,1995), apresenta-se uma ilustração de grupo aberto e grupo fechado.

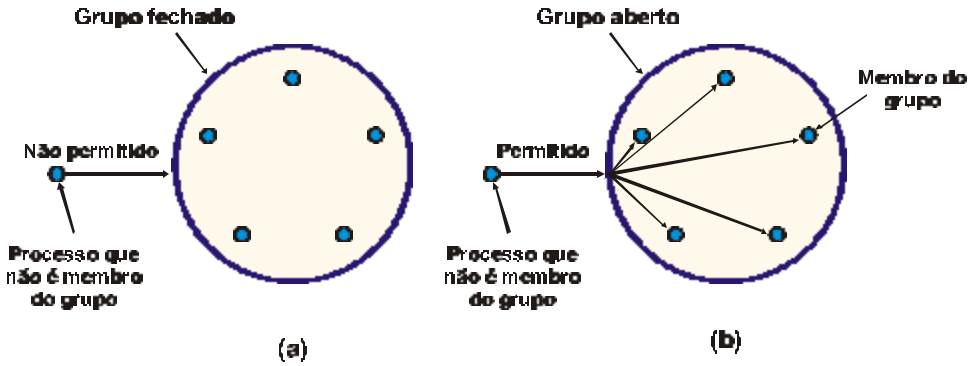

Figura 21 (a) Grupo Fechado (b) Grupo Aberto

A decisão de usar grupo fechado ou grupo aberto, normalmente está relacionada com o tipo de aplicação que estará sendo executada, usando estes mecanismos de comunicação em grupo. Se for uma aplicação típica de processamento paralelo, uma coleção de processos trabalhando juntos para a solução de um problema, neste caso não interessa outro processo de fora interagir com algum processo do grupo, sendo típico para um grupo fechado. De outro lado, em uma aplicação de servidores replicados, um processo não membro do grupo, um cliente, pode enviar uma mensagem para o grupo fazendo alguma requisição aos servidores.

Ainda quanto à organização temos grupos de semelhantes e grupos hierárquicos. Esta característica está relacionada à estrutura interna do grupo. Em alguns grupos os processos são iguais. Nenhum processo é gerente e todas as decisões são feitas coletivamente. Em outros grupos existe um processo que é o coordenador e os demais

fazem o que é determinado por este coordenador. Neste caso quando uma requisição é feita, seja por um cliente externo ou por um dos membros do grupo, a requisição é enviada ao coordenador. O coordenador decide qual dos processos do grupo é o mais indicado para executar esta solicitação. Este padrão de comunicação está ilustrado na Figura 22 (TANENBAUM ,1995).

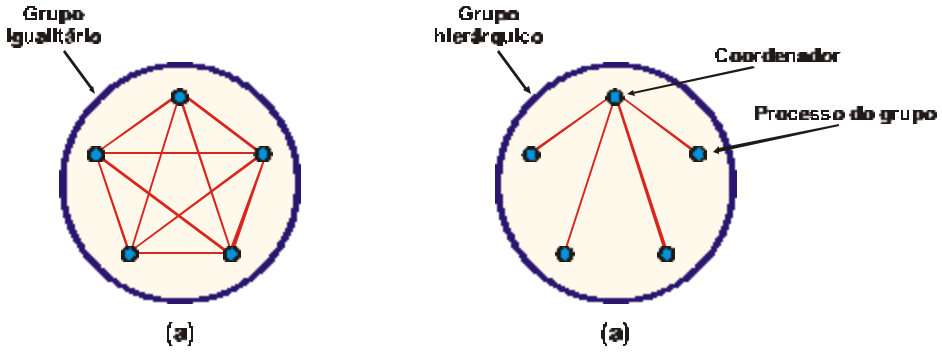

Figura 22 (a) Comunicação em Grupo de Semelhantes
(b) Comunicação em Grupo Hierárquico.

Cada uma destas organizações tem suas próprias vantagens e desvantagens. Na comunicação em grupos de semelhantes não tem um ponto único de falha. Se um dos processos falhar, o grupo continua, apenas se torna menor. A desvantagem é que a tomada de decisão é mais complicada. Para alguma decisão, tem que envolver todos do grupo, gerando uma demora e um *overhead*. No grupo hierárquico temos o oposto sendo as decisões tomadas pelo coordenador. Qual processo do grupo vai executar qual atividade e quando um processo termina, ele avisa ao coordenador que está pronto para outra atividade. A perda do coordenador em um grupo hierárquico causa uma parada nas atividades do grupo.

3.3 Gerenciamento na Comunicação em Grupos

No mecanismo de comunicação em grupo é necessário um controle sobre a criação e destruição de grupos, bem como permitir que processos se integrem ao grupo ou deixem o grupo. Um enfoque é ter um servidor de grupo para o qual todas as requisições possam ser feitas. Este servidor de grupo pode manter um controle total de todos os seus grupos e quais participantes de grupo pertencem a qual grupo. Este

método é fácil de entender, eficiente e de fácil implementação. Infelizmente ele compartilha com a maior desvantagem das técnicas centralizadas, um único ponto de falha. Se o servidor de grupos falhar, o gerenciamento do grupo deixa de existir. Provavelmente a maioria dos grupos terá que ser reconstruídos, e certamente interrompendo o trabalho que estava em andamento.

O enfoque oposto é gerenciar os grupos de uma forma distribuída. Em um grupo aberto, um processo de fora do grupo, pode enviar uma mensagem para todos do grupo anunciando a sua presença. Em um grupo fechado, ao menos com relação à solicitação de participar do grupo, também será enviada uma mensagem para todos. Este grupo fechado, só aceita uma mensagem de algum processo fora do grupo, quando esta mensagem é de solicitação para participar do grupo. Uma mensagem direta sem antes estar participando do grupo fechado, não seria aceita. Para um processo deixar de fazer parte de um grupo, basta enviar uma mensagem de adeus a todos do grupo.

Consideram-se ainda duas características relativas ao controle dos participantes do grupo. Se o membro do grupo falhar ele efetivamente deixa de pertencer ao grupo. O problema é que pode não existir um aviso a todos os outros membros do grupo, como quando o processo deixa o grupo de forma padrão. Os demais membros têm que descobrir por conta própria que aquele participante não está respondendo e o excluir do grupo. Outro ponto é o sincronismo necessário entre as mensagens sendo enviadas e a entrada ou saída de um membro no grupo. Todo processo que for integrado ao grupo no momento t deverá receber a partir deste momento t, todas as mensagens enviadas. Da mesma forma quando deixa o grupo no momento z, todas as mensagens enviadas após z já não são mais endereçadas a este processo que deixou o grupo. Uma forma de resolver esta situação é transformar a operação de inclusão no grupo ou retirada do grupo em uma mensagem especial de inclusão ou retirada que será transmitida de forma síncrona na seqüência das demais mensagens para o grupo. Uma última preocupação relacionada a controle dos participantes do grupo é quando muitas máquinas param de forma anormal, por exemplo quando ocorre um particionamento da rede, deixando o grupo inoperante. Algum protocolo é necessário para reconstruir o grupo com as máquinas que restaram.

Para que um processo envie uma mensagem a um grupo é necessária uma forma de identificar o grupo. Uma forma é atribuindo a cada grupo um endereço único assim como temos um número único de identificação para cada processo. Se a rede suportar *multicast*, o endereço do grupo pode ser associado com o endereço de *multicast*, assim

cada mensagem enviada ao grupo pode ser direcionada especificamente para aquele grupo identificado pelo endereço de *multicast* e só vai ser recebida pelas máquinas que fazem parte daquele endereço de *multicast.* Se o hardware não suporta *multicast* mas aceita *broadcast*, as mensagens podem ser enviadas por *broadcast*. Cada *kernel* receberá a mensagem e vai avaliar o endereço do grupo. Se algum processo que está rodando nesta máquina faz parte do grupo ele recebe a mensagem, caso contrário, a mensagem será descartada. Em ultimo caso se nem *multicast* nem *broadcast* for suportado na rede, o *kernel* na máquina que envia deverá ter uma lista de todas as máquinas que tem processos pertencentes a este grupo da mensagem a ser enviada. O *kernel* então envia em uma conexão ponto-a-ponto a cada um dos participantes do grupo uma de cada vez, a mensagem a ser transmitida. Estas três implementações são vistas na Figura 23 onde um processo 0 envia uma mensagem para um grupo consistindo dos processos 1, 3 e 4.. Um ponto importante é que mesmo nos três casos o processo que envia manda apenas uma mensagem para o endereço do grupo e a mensagem vai para todos os membros do grupo. A forma como será transmitida é problema do sistema operacional. O processo que envia não tem que se preocupar com o tamanho do grupo ou se a comunicação vai ser por *multicasting*, *broadcasting* ou *unicasting*.

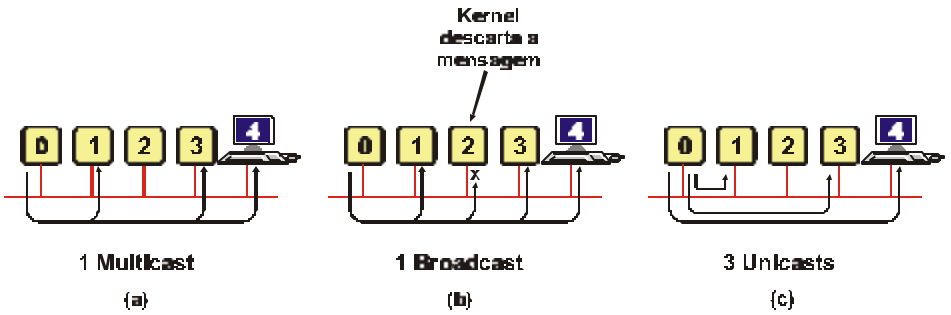

Figura 23 (a) *Multicasting* (b) *Broadcasting* (c) *Unicasting*

Um segundo método de endereçamento do grupo é quando o processo do usuário final que envia tem uma lista explicita de todos os destinos, por exemplo, endereços IP. Quando este método é usado, um parâmetro na chamada de envio da mensagem é um ponteiro para uma lista de endereços. Este método tem um sério inconveniente que é forçar o processo membro do grupo de saber precisamente quem faz parte do grupo. Assim deixa de ser transparente. Sempre que existir uma mudança de participante no

grupo o processo do usuário final precisa atualizar sua lista de membros do grupo. No exemplo da Figura 23 esta administração é feita pelo *kernel* e o processo do usuário final não se preocupa com isto. Um terceiro método de endereçamento é chamado de endereçamento com predicado. Neste sistema cada mensagem é enviada para todos os membros do grupo porém contendo um predicado, uma expressão booleana, para ser avaliada. Este predicado pode envolver o número da máquina receptora, suas variáveis locai ou outros fatores. Se este predicado após ser avaliado for verdade então a mensagem é aceita.

Seria ideal se fosse possível ter um conjunto de primitivas comuns, tanto para comunicação ponto-a-ponto como para comunicação em grupo. Quando usa-se RPC tem-se o envio e a recepção de mensagens usando duas simples primitivas tais como "send" e "receive", respectivamente. Para a comunicação em grupo uma mensagem enviada pode gerar diversas respostas. Se for possivel usar uma biblioteca de procedimentos poderemos ter uma semântica única para envio de mensagens, com dois parâmetros. Se fosse RPC um parâmetro era endereço do processo e o segundo endereço da mensagem. Se for para grupo o primeiro parâmetro é o endereço do grupo e segundo endereço da mensagem. A chamada pode ser usando buffer ou não, bloqueada ou não, confiável ou não tanto para ponto-a-ponto como para grupo. Em geral estas escolhas são feitas pelos projetistas do sistema e ficam sem flexibilidade de parametrização. De maneira similar o "receive" indica a intenção de receber uma mensagem, podendo o processo ficar bloqueado até o recebimento da mensagem. Se usarmos a mesma semântica o "receive" deveria funcionar para comunicação ponto-a-ponto e para grupos. Devido a estas dificuldades para manter a mesma semântica com formas tão diferentes de processamento, alguns projetistas incluíram novas primitivas como "group_send" e "group_receive". Se necessário associar a resposta a uma chamada podemos usar "getreply" onde após o envio de uma mensagem podemos chamar "getreplay" repetidamente para coletar todas as respostas associadas.

3.4 Propriedades da Comunicação em Grupos

Uma propriedade importante no mecanismo de comunicação em grupo, é a propriedade do tudo-ou-nada. A maioria dos sistemas que implementam os mecanismos de comunicação em grupo, são projetados para que as mensagens enviadas a um grupo

cheguem corretamente a todos os membros deste grupo ou não cheguem a nenhum deles. Esta propriedade do tudo-ou-nada é também conhecida como atomicidade ou *broadcast* atômico. A atomicidade é importante pois torna mais fácil a programação de um sistema distribuído. Quando qualquer processo enviar uma mensagem para um grupo ele não deve preocupar-se se algum outro membro do grupo não recebeu esta mensagem. Com esta propriedade da atomicidade o processo que envia a mensagem para um grupo irá receber uma mensagem de erro se um ou mais integrantes do grupo teve problema no recebimento sendo que os demais vão ignorar esta mensagem. Do ponto de vista de programação, tudo se passa como se fosse enviado para apenas um receptor.

Outra propriedade importante é o ordenamento das mensagens. Além do conceito tradicional de envio e recebimento de mensagens, quando tratamos de comunicação em grupo devemos ter em mente o conceito de entrega da mensagem. Assim uma mensagem recebida para um destino final passa por um processo de entrega que eventualmente precisa ser controlado para que a seqüência de entrega seja a mesma seqüência de envio. Alguns algoritmos que tratam estas características de ordenamento em ambiente de **multicast** podem ser encontrados em Coulouris, 2001 no capítulo 11.

Diretório Mensagens do Usuário U3

Seq	Usuário	Assunto
1	U1	Novo modelo?
2	U2	Re: Novo modelo ?
3	U5	Outro assunto

Diretório Mensagens do Usuário U4

Seq	Usuário	Assunto
1	U2	Re: Novo modelo ?
2	U5	Outro assunto
3	U1	Novo modelo?

Figura 24 Diretório de Mensagens dos Usuários U3 e U4

Para exemplificar esta propriedade de ordenamento de mensagens, considera-se uma situação que ocorre em um grupo de assinantes de uma lista de discussão na Internet. Uma lista de assinantes pode ser considerada como um grupo onde todos os membros do grupo estão interessados no assunto que é tratado na lista. Assim quando alguém envia uma mensagem para o grupo todos recebem esta mensagem e alguns

podem responder a referida mensagem, resposta esta que também será propagada para todos os membros do grupo. Observa-se o diretório de mensagens recebidas de dois usuários deste grupo, usuário U3 e usuário U4, conforme Figura 24. O usuário U3 recebe uma mensagem com o assunto "novo modelo?" enviada por U1 e logo em seguida uma outra mensagem "Re: novo modelo?" enviada por U2 onde uma resposta para a pergunta anterior estava sendo fornecida. Se observarmos o diretório de mensagens do usuário U4 veremos que este primeiro recebeu a resposta "Re: novo modelo?" , uma outra mensagem enviada por U5 e depois é que recebeu a pergunta "novo modelo?" caracterizando uma troca de ordem nas mensagens recebidas. Esta troca ocorreu pois o servidor de mensagens trabalha de forma assíncrona e a entrega das mensagens não tem nenhuma garantia de ordenamento. Certamente a mensagem enviada por U1 com a pergunta "novo modelo?" não pode ser entregue na primeira tentativa e quando foi reenviada posteriormente para o usuário U4 este já tinha recebido antes a resposta da pergunta.

Um mecanismo de comunicação em grupo precisa ter uma semântica bem definida com relação à ordem em que as mensagens serão entregues. A melhor garantia para isto é fazer com que as mensagens sejam entregues na mesma ordem em que foram enviadas. Neste exemplo de lista de mensagens anterior esta garantia não é implementada pois o custo seria muito grande e os sistemas de lista se valem dos recursos já existentes de envio de mensagens via Internet e que não contemplam ordenamento.

A ordenação de mensagens é um ponto muito importante na comunicação em grupo e assim vamos detalhar mais este assunto baseado nos conceitos de ordenação de mensagens em serviços de *broadcast* (BROADCAST, 1997), (ATTIYA, 1998), (MULLENDER, 1993), e também no sistema AMOEBA (KAASHOEK, 1993). Existem quatro tipos diferentes de ordenação que normalmente estão implementadas nos mecanismos de comunicação de grupo:

- **Sem Ordem** – Onde as mensagens são enviadas ao grupo sem a preocupação de ordenamento. Tem o menor *overhead* pois não necessita controle algum de seqüência porém pode não ser adequado para muitas aplicações.
- **Ordenamento FIFO** – Garante que todas as mensagens de um membro sejam entregues aos demais membros do grupo na ordem em que elas foram enviadas. Para todas as mensagens M1 e M2 e todos os processos Pi

e Pj, se Pi envia M1 antes de enviar M2, então M2 não é recebida em Pj antes que M1 seja recebida.

- **Ordenamento CAUSAL** – No ordenamento causal introduz-se o conceito de mensagem dependente de outra. Um exemplo para ilustrar, é quando se recebem mensagens de uma lista na Internet. Um membro da lista envia uma pergunta para a lista, esta mensagem é enviada a todos os membros da lista. Na seqüência alguém responde a solicitação e esta resposta também é enviada a todos. Na caixa postal de um participante da lista chega primeiro a mensagem com a resposta e mais tarde a mensagem com a pergunta. No ordenamento causal as mensagens estão em ordenamento FIFO e se um membro após receber a mensagem A envia uma mensagem B, é garantido a todos os membros do grupo que vão receber A antes de B. Para todas as mensagens M1 e M2 e cada processo Pi, se M1 acontece antes de M2 então M2 não é recebida em Pi antes que M1 seja. A Figura 25 apresenta um exemplo gráfico de ordenamento causal. Nesta execução duas mensagens são controladas em termos de entrega. Primeiro a mensagem de p1 (0,2,0) é atrasada até que a mensagem anterior (0,1,0) chegue vinda de p1. Depois a mensagem (1,3,0) vinda de p0 é atrasada até que a mensagem (0,3,0) vinda de p1 (que aconteceu antes) tenha chegado.

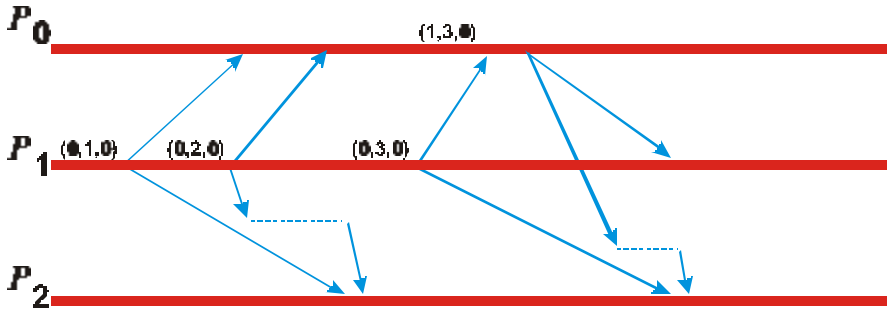

Figura 25 Ordenamento Causal

- **Ordenamento Total** – No ordenamento total cada membro do grupo recebe todas as mensagens na mesma ordem. Para todas as mensagens M1 e M2 e todos os processos Pi e Pj, se M1 é recebida em Pi antes que M2 seja, então M2 não é recebida em Pj antes que M1 seja. Este ordenamento

total é o melhor de todos aqui apresentados e torna a programação fácil, mas é difícil de ser implementado.

Uma propriedade de sobreposição de Grupos está associada ao fato que um processo pode ser membro de diversos grupos ao mesmo tempo. Esta situação pode levar a que se tenham inconsistências. Pode-se observar na Figura 26 que mostra dois grupos 1 e 2. Os processo A, B e C são membros do grupo 1. Os processo B,C e D são membros do grupo 2. Supondo que os processos A e D decidam simultaneamente enviar uma mensagem a seus respectivos grupos, e que o sistema use a ordenação em tempo global dentro de cada grupo. Considerando que se usa *unicast*, a ordem das mensagens é mostrada na Figura 26 (TANENBAUM ,1995), através da numeração de 1 a 4. Novamente temos uma situação em que dois processos B e C, recebem mensagens em ordem diferente. O processo B primeiro recebe uma mensagem de A, seguida por uma vinda de D. O processo C recebe na ordem oposta. O problema é que embora tenhamos ordenação em tempo global, dentro do grupo, não há necessariamente qualquer coordenação entre os diversos grupos. Alguns sistemas suportam uma ordenação no tempo muito bem definida entre grupos sobrepostos, mas outros não.

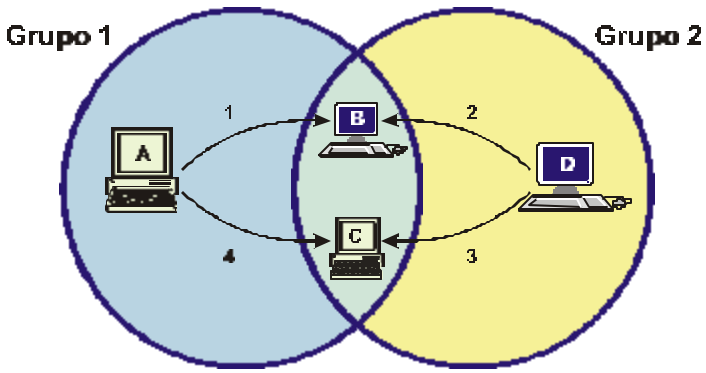

Figura 26 Quatro Processos A,B,C e D e Quatro Mensagens.
Os processos B e C recebem as mensagens de A e D em ordem diferente

Outra propriedade importante nos mecanismos de comunicação em grupo se refere a escalabilidade. Muitos algoritmos funcionam bem enquanto os grupos possuem poucos elementos e não houver muitos grupos. Podem-se ter problemas se o grupo tiver centenas ou milhares de componentes, ou se tivermos milhares de grupos e se o sistema for muito grande.

4 MECANISMOS DE COMUNICAÇÃO EM GRUPO

Existem atualmente diversos mecanismos de comunicação em grupo sendo que alguns foram desenvolvidos no início da década de 90 e outros mais recentes inclusive usando tecnologia de orientação a objetos.

Um ponto importante a ser considerado com relação à rede onde vai rodar o mecanismo de comunicação em grupo é a possibilidade de particionamento da rede. Podemos ter redes locais independentes, redes locais interligadas via WAN ou grandes redes interligadas formando uma grande WAN. A seguir apresentamos resumidamente alguns mecanismos de comunicação em grupo conforme sua classificação.

4.1 Mecanismos de Partição Primária

Uma partição é onde a comunicação em grupo ocorre somente entre os membros desta rede. Mecanismos de comunicação de grupo que são voltados exclusivamente para partições únicas são também chamados de mecanismos de partição primária. Alguns mecanismos que não aceitam o particionamento são voltados especificamente para redes locais onde as características de confiabilidade, latência, taxa de erro permitem a adoção de algoritmos simplificados e conseqüentemente com melhor performance do que aqueles que aceitam particionamento. A seguir alguns exemplos de mecanismos de partição primária.

ISIS

O ISIS (BIRMAN, 1994), é composto de uma serie de ferramentas na forma de uma biblioteca de procedimentos ligados diretamente à aplicação. Desenvolvido na Universidade de Cornell no ano de 1985 pelo professor Birman, (2002), provê funcionalidades para a criação dinâmica de grupos de processos, participar de um grupo, e mandar mensagens aos processos do grupo com várias garantias de ordenamento. O sincronismo das mensagens, FIFO, ordenamento causal e ordenamento total são permitidos com primitivas de *multicast.* Este conjunto de software suporta sincronismo virtual e tolerância a falha. Este trabalho foi descontinuado e em seu lugar o autor passou a pesquisar um novo sistema chamado de HORUS (RENESSE, 1996).

PHOENIX

PHOENIX (MALLOTH, 1996), foi desenvolvido no Swiss Federal Institute of Technology em Lausanne. É um kit de ferramentas de comunicação de grupo que trabalha com membros de grupo em partição primária e sincronismo de mensagens de uma única visão em ambientes assincronos. PHOENIX implementa roteamento dinâmico, canais de comunicação confiáveis com orientação a datagrama, primitivas de comunicação de grupo e ordenação de mensagens. Suporta um alto número de processos designando a eles diferentes regras: membros principais, clientes e receptores. Membros principais gerenciam estados compartilhados e tem forte garantia de confiabilidade com respeito à entrega de mensagens e alterações nos membros do grupo. Comunicam-se uns com os outros usando grupo *multicast.* Clientes interagem com membros principais enviando requisições para eles e recebendo respostas e alterações nos participantes do grupo. Finalmente os receptores somente recebem informações difundidas pelos membros do grupo. A interface de programação PHOENIX prove a abstração de orientação a objeto para suportar grupos de objetos ao invés de grupos de processos.

RMP

O protocolo *multicast* confiável (RMP – Reliable Multicast Protocol) (WHETTEN, 1994), provê serviços de *multicast* atômico, confiável e totalmente ordenado a partir de serviços de datagrama *multicast* não confiáveis tais como o IP *multicasting*. RMP provê muitas opções de comunicação, incluindo sincronismo virtual, um modelo publicante/assinante de entrega de mensagens, um modelo cliente/servidor, um serviço de nomes implícito, manuseadores mutuamente exclusivo para mensagens e mutuamente exclusivos controles de *lock*. RMP é baseado em técnicas de *token RING.* Este protocolo prove *broadcast* atômico totalmente ordenado para um grupo único de clientes em uma LAN.

4.2 Mecanismos Particionáveis

Alguns mecanismos de comunicação em grupo são chamados de particionáveis pois podem tratar possíveis particionamentos que possam ocorrer na rede. Uma partição

é onde a comunicação em grupo ocorre somente entre os membros desta rede. Assim um mecanismo que trata particionamento, pode controlar um grupo formado por duas LANs interligadas e caso ocorra uma queda no *link*, ira separar os processos em dois grupos e posteriormente quando restabelecer o *link* fará uma intercalação dos dois grupos, voltando ao status do grupo original. A seguir apresenta-se resumidamente alguns mecanismos de comunicação em grupo do tipo particionavel.

TOTEM

TOTEM (MOSER, 1996), foi desenvolvido na Universidade da Califórnia em Santa Bárbara e é um conjunto de protocolos de comunicação para a construção de sistemas distribuídos tolerantes a falha. TOTEM é voltado para aplicações complexas nas quais confiabilidade e performance em tempo real são críticas. O sistema TOTEM manuseia falhas e recuperação de processos, particionamento de rede e reagrupamento de redes partidas. Mesmo que ocorra um particionamento de rede as operações continuam em cada parte da rede. Além disso, TOTEM fornece diversas primitivas de *multicast* com ordenação para os grupos de processos, com alta performance e baixa latência prevista. Este protocolo explora as capacidades de *multicast* existentes no hardware em redes LAN e a localização dos grupos de processos de forma a prover garantias de tempo real ao software.

HORUS

O projeto HORUS (RENESSE, 1996), foi originalmente lançado com um esforço de redesenhar a série de ferramentas ISIS em um conjunto de pequenas, unidades modulares bem definidas. HORUS provê um modelo de sincronismo virtual que permite que uma aplicação faça progresso mesmo em uma partição minoritária, indicando para a aplicação se ela no momento é parte da partição primaria ou não. Uma das mais interessantes características do HORUS reside na sua arquitetura de comunicação que trata o protocolo como um tipo de dados abstrato. Camadas de protocolo podem ser empilhadas no topo de cada uma delas em uma variedade de formas em tempo de execução. HORUS provê um ambiente de composição de protocolo que permite protocolos personalizados possam ser compostos a partir de outros existentes. Esta arquitetura tem a vantagem adicional que uma aplicação somente paga propriedades que usa.

ENSEMBLE

Recentemente uma nova geração do sistema HORUS tem aparecido e se chama ENSEMBLE (HAYDEN, 1998). ENSEMBLE é uma arquitetura de protocolo de alta performance que permite reconfiguração de redes e voltada para aplicações adaptativas. A funcionalidade básica do ENSEMBLE é controlar os participantes do grupo e prover suporte de comunicação entre os membros do grupo e comunicação que vem de outros não membros do grupo. Dependendo da natureza da aplicação o usuário pode trabalhar com ENSEMBLE diretamente, como uma ferramenta suportando replicação de dados, colaboração, ou coordenação. Contudo ENSEMBLE pode também ser usado para controlar outras tecnologias, tais como o resultado final de comunicação padrão sockets em camadas sobre IP.

TRANSIS

O sistema de comunicação TRANSIS (DOLEV, 1996), desenvolvido na Universidade Hebrew de Jerusalém, suporta o processamento de grupos de comunicação com diversas formas de operação de *multicast* em grupo. Ordenação FIFO, ordenação causal, ordenação total e entrega segura. TRANSIS tem um protocolo para entrega de mensagens confiáveis que otimiza a performance para o hardware existente na rede e também aceita particionamento de rede. Recentemente o grupo do TRANSIS tem desenvolvido protocolos para escalar gerenciamento de membros do grupo em redes WAN.

NEWTOP

É um protocolo de comunicação em grupo de propósito geral (EZHILCHELVAN, 1995). Assume que processos podem pertencer a diversos grupos simultaneamente, o tamanho do grupo pode ser grande e os processos podem estar geograficamente espalhados e distantes comunicando-se pela internet. Em ambiente de comunicação assíncrona é onde temos dificuldade de medir de forma precisa o tempo de transmissão das mensagens, e a rede envolvida pode ser particionada, impossibilitando que todos os processos possam se comunicar entre si. NEWTOP pode prover ordenação causal e ordenação total na entrega das mensagens aos membros de um grupo e também assegurando que a ordenação total é preservada para processos em multi grupo. Tanto

protocolo de ordenamento simétrico como assimétrico são suportados, permitindo que um processo possa ter versão simétrica em um grupo e versão assimétrica em outro grupo. NEWTOP é dinâmico e tolerante a falha: ordenamento e controle de vida são preservados mesmo se mudanças nos membros do grupo ocorrem devido às falhas em processos, saída voluntária de processos ou formação de novos grupos.

ATOMIC

O sistema ATOMIC (KOCH, 2000), é um sistema de comunicação em grupo que foi projetado para redes do tipo ATM. O protocolo usado neste sistema cria sua própria topologia para a distribuição dos dados. Os processos são organizados em grupos locais onde inicialmente é feita a ordenação do grupo. A ordenação através de outros grupos ocorre quando processos passam a pertencer a mais de um grupo.

Sendo altamente escalar, ATOMIC permite milhares de nós distribuídos sobre uma área fisicamente grande, formando um sistema único, que contém um grande número de processos. O protocolo foi projetado para prover alto desempenho e baixa latência na entrega das mensagens. Baseado em controles de tempo (*timestamps*) derivados de relógios físicos ou lógicos.

O sistema de comunicação em grupo ATOMIC, se vale de um protocolo *multicast* confiável baseado em números de seqüência para conseguir entregar as mensagens de forma confiável e controles de tempo para obter ordenamento total. O protocolo é iniciado pelo receptor, o que prove uma situação de latência muito baixa no caso de perda de mensagens.

4.3 Comparativos

Considerando os produtos acima descritos pode-se classificar os mesmos segundo a sua orientação para uso em LAN ou WAN, conforme Quadro 3.

Quadro 3 Mecanismos de Comunicação em Grupo e Ambientes de Rede

LAN	WAN
ISIS	TOTEM
PHOENIX	HORUS
RMP	ENSEMBLE
	TRANSIS
	NEWTOP
	ATOMIC

Neste trabalho três mecanismos foram selecionados para um estudo mais aprofundado sendo dois especificamente em ambiente JAVA, INTERGROUP (BERKET, 2000), e JGROUP (MONTRESOR, 2000), e o terceiro fortemente baseado em ambiente UNIX, porém com versões para outras plataformas, o SPREAD (AMIR, 1998). A razão da escolha destes três mecanismos refere-se ao fato de todos terem uma documentação mais detalhada para análise e facilidade na obtenção do software correspondente para a realização de experimentos práticos. No produto INTERGROUP estavam disponíveis apenas as classes sem o código fonte. No produto JGROUP além das classes para execução também estava disponível o código fonte com todos os arquivos do tipo fonte JAVA. No terceiro mecanismo de comunicação em grupo selecionado, o SPREAD, além de uma completa documentação, versões compiladas para diferentes plataformas também uma copia completa de todos os fontes estavam disponíveis via Internet. Considerando que este mecanismo SPREAD poderia ser empregado em diferentes plataformas de hardware como Intel, PowerPc, Risc e diferentes sistemas operacionais como UNIX, WINDOWS, Mac OS e ambiente JAVA, este foi o mecanismo escolhido para ser usado na avaliação de desempenho e aplicação de demonstração que é descrita no capítulo 8.

Estes três mecanismos serão objetos de análise mais detalhada nos capítulos seguintes.

5 INTERGROUP

O protocolo INTERGROUP, resultado de uma dissertação de Berket (2000), submetida à Universidade da Califórnia – Santa Bárbara em dezembro de 2000. É voltado para o ambiente de Internet, apresentando escalabilidade em ambiente de rede com alta latência e grande número de nós. O protocolo INTERGROUP trata o problema da escalabilidade sobre diferentes enfoques. Ele redefine o conceito de membros do grupo, permitindo alterações voluntárias dos membros do grupo, adicionando uma seleção orientada a recepção de garantias de entrega que permite heterogeneidade no conjunto de receptores e prove serviços confiáveis em ambientes de larga escala. O sistema INTERGROUP compreende vários componentes executando em vários equipamentos formando um sistema único. Cada componente é responsável por parte dos serviços necessários para implementar um sistema de comunicação em grupo para redes WAN. Os componentes podem ser categorizados como: (1) controle de hierarquia, (2) *multicast* confiável, (3) envio e entrega de mensagens e (4) controle de processos membros do grupo. A implementação do protótipo do INTERGROUP foi feita em JAVA e os testes de performance quando do desenvolvimento foram feitos tanto em redes LAN como WAN.

5.1 Características

O protocolo INTERGROUP se propõe a diminuir o custo dos algoritmos de reparação necessários para o controle dos membros do grupo usando as seguintes estratégias. No sistema INTERGROUP nem todos os processos são considerados iguais. Em cada processo do grupo ele é classificado segundo a sua atividade mais recente. Se o processo envia dados para o grupo recentemente é classificado como um emissor ativo. Emissores ativos são membros do grupo emissor no processamento em grupo. Somente os emissores pertencentes a este grupo emissor necessitam participar nos algoritmos de decisão por consenso. Também são usados mecanismos voluntários para entrar ou sair de um grupo. Estes mecanismos voluntários têm a vantagem de não exigir a execução dos algoritmos de reparo no controle de membros do grupo.

Também avança além dos conceitos tradicionais relativos a escolha do serviço de entrega das mensagens provendo mais flexibilidade para a aplicação e reduzindo os

custos dos algoritmos de reparação no controle de membros do grupo. No sistema INTERGROUP a entrega de mensagens que vem de um processo do grupo é determinada pela aplicação, no lado do receptor. Cada processo pode escolher diferentes tipos de entrega. Os seguintes serviços de entrega estão disponíveis:

- Não confiável e não ordenada – Mensagens recebidas pelo grupo são entregues diretamente a aplicação. Algumas mensagens podem não ser recebidas e também múltiplas cópias da mesma mensagem podem ser recebidas. Além disso não há garantia na ordem em que as mensagens são recebidas. É a funcionalidade básica provida pelo IP *multicast*.
- Confiável ordenada na fonte – Todas as mensagens de uma fonte em particular serão recebidas pela aplicação (a menos que tenha uma falha no processo) e elas serão entregues ordenadas em numeração seqüencial. Este serviço é bem adequado para aplicações tipo transferência de arquivos em *multicast* e outras aplicações que usam TCP/IP. O serviço fornecido é similar ao TCP/IP.
- Grupo confiável ordenada por tempo – Mensagens são recebidas pela aplicação em uma ordenação baseada em tempo considerando todo o grupo. As possíveis alterações de membros do grupo que possam ocorrer são controladas de forma a assegurar que as mensagens recebidas pelas aplicações obedeçam a um sincronismo virtual. Este serviço está próximo da idéia de mensagens com acordo nos sistemas de comunicação em grupo.

O controle da informação no sistema INTERGROUP é comunicado através do grupo de controle. O grupo de controle consiste de exatamente um processo de controle em um determinado equipamento de todo o conjunto. Um processo de controle roda independentemente dos outros processos em um equipamento. Cada processo em um equipamento envia e recebe informações de controle via o processo de controle em seu equipamento. Um processo de controle não interage diretamente com a aplicação. Os processos de controle são organizados em uma estrutura hierárquica auto-organizável.

O sistema INTERGROUP é composto de diversos componentes, executando em um mesmo equipamento. Cada componente é responsável por parte do serviço necessário para construir um sistema de comunicação em grupo para ambiente WAN. Os componentes podem ser categorizados como:

- Controle de hierarquia – Usado para a troca de informações de controle entre os equipamentos no sistema. Cada equipamento tem um processo de controle que é responsável pelas informações de controle para todos os outros processos neste equipamento. O controle de hierarquia também provê mecanismo para determinar a estabilidade de mensagens, e o mecanismo para buscar uma forma fraca de consenso para o grupo.
- *Multicast* confiável – Mecanismo para requisitar a retransmissão de mensagens, mecanismo para retransmitir as mensagens e para detectar se uma mensagem precisa ser recuperada.
- Distribuição e entrega de mensagens – Executa o controle de fluxo nas mensagens enviadas para um grupo, ordena e entrega mensagens para a aplicação baseada na escolha pela aplicação do serviço de entrega e detecta mensagens faltantes.
- Processamento de controle dos membros do grupo – Mantém uma visão de todos os participantes membros do grupo. Esta visão depende do serviço de entrega selecionado pela aplicação e se o processo é membro do grupo que pode enviar.

5.2 Implementação

O INTERGROUP está implementado como três componentes principais. O controle de hierarquia, *multicast* confiável e ordenamento e entrega de mensagens. Cada componente é construído usando um conjunto de módulos do tipo thread que se comunicam via um modelo de evento/mensagem assíncrono. Como diferentes serviços requerem diferentes módulos, cada componente é projetado de modo que módulos e fluxo de mensagens possam ser dinamicamente construídos e mantidos. Eles são construídos em volta da classe "*InterGroupThreadedModule*" (IGTM). A classe IGTM implementa duas interfaces: Runnable e MessagePublisher (MP). Também contém uma classe PriorityQueue que implementa a interface MessageSubscriber (MS). A interface Runnable é definida na plataforma standard JAVA 2 v1.2.2 API. Permite que IGTM execute como uma thread.

A interface MP permite para a publicação de eventos/mensagens para classes que implementam a interface MS. As subscrições são feitas via método *attach*. A interface MP também prove mecanismos para cancelar a subscrição usando método *detach*. A

implementação da interface MP na classe IGTM usa uma estrutura de dados que mapeia o tipo de mensagem para MS. Sempre que uma mensagem é publicada via o método notify, o método update é chamado para cada MS que usa este tipo de mensagem. Esta implementação permite que assinantes assinem e recebam eventos e mensagens de diferentes tipos e permite múltiplos assinantes a receber o mesmo tipo de eventos e mensagens.

A classe PriorityQueue implementa a interface MS. Recebe mensagens via o método update e guardam elas em um buffer até que a classe IGTM esteja pronta para processar elas. Quando IGTM está pronta para receber as mensagens da fila, elas são escolhidas baseadas em um algoritmo que considera a prioridade das mensagens na fila. Desta forma não temos garantia que mensagens de diferentes prioridades deixam a fila na mesma ordem em que chegaram na fila. Contudo mensagens de mesma prioridade se comportam como FIFO. As propriedades da classe IGTM permitem fácil ajuste de fluxo de dados dinâmicos, assíncronos e configuráveis entre os componentes.

5.3 Resultado Prático da Instalação

Este mecanismo de comunicação de grupos, INTERGROUP, está disponível em sua pagina na Internet para ser recebido via Internet. (BERKET, 2000), É fornecido apenas o conjunto de classes em formato binário para ser executado em uma máquina JAVA. O código fonte não é fornecido. Um pequeno exemplo de uso, das classes com chamadas básicas para envio de mensagem a grupos, está disponível junto com o aplicativo. Outro exemplo mostra uma aplicação de troca de mensagens entre usuários, usando os conceitos de comunicação em grupo onde cada usuário pode enviar mensagens a todos os membros do grupo.

A documentação do INTERGROUP se restringe a alguns artigos e a própria dissertação escrita pelo autor do produto. Junto com os arquivos binários apenas dois arquivos textos muito simplificados com informações para instalação e exemplo muito básico de uso do sistema.

Após instalado o produto, foram feitos testes com os programas de demonstração que o acompanham. Uma das aplicações permite o uso do INTERGROUP junto com o aplicativo, na mesma execução mas neste caso só uma instância do aplicativo é permitida. Para testar o aplicativo de troca de mensagens foi ativada uma instância do

servidor INTERGROUP no WINDOWS e mais 5 clientes para troca de mensagens sendo dois clientes no LINUX e três clientes no WINDOWS. Como tanto o servidor como os clientes estão codificados em JAVA, a performance do sistema apresenta uma lentidão muito grande na execução. Só para montar a tela gráfica cada chamada do cliente leva em torno de 30 segundos. Cabe ressaltar que quando foi configurado para que o servidor INTERGROUP rodasse no LINUX, pensando em uma melhor performance, o mesmo não funcionou impossibilitando este teste. Na Figura 27 apresentam-se as telas das três aplicações clientes de troca de mensagens que estavam rodando no WINDOWS.

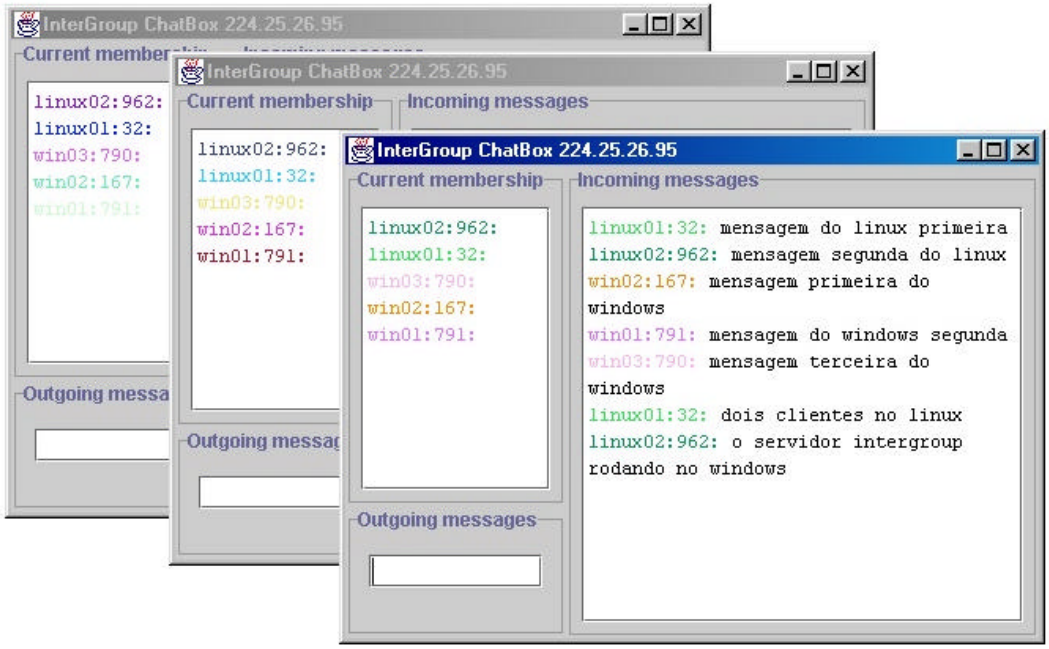

Figura 27 Exemplo do Aplicativo de Troca de Mensagens Usando INTERGROUP

Este software tem evoluído pois no início deste trabalho a documentação era mínima e agora no final, passado alguns meses, ao concluir a avaliação prática, encontramos uma nova versão de documentação, com mais exemplos e com mais detalhes quanto ao uso do INTERGROUP. Entretanto, como tudo está em JAVA temos a vantagem da portabilidade mas de outro lado, a performance deixa a desejar para aplicações que requerem uma maior volume de transações. Atualmente, o software está na versão 1.0 alpha e apresenta algumas deficiências já bem conhecidas tais como:

- Problemas de memória quando o nó INTERGROUP não libera memória de forma correta. Alguns objetos não deixam de ser referenciados corretamente causando falta de memória ao sistema.
- Após tratar um grande número de grupos, maior que 100, incluindo e excluindo estes grupos o nó INTERGROUP deixa de responder. Isto ocorre porque o nó INTERGROUP usa a classe JAVA.útil. Timer para programar a execução de algumas *tasks* e a mesma classe é usada para todas as *tasks* que tratam todos os grupos onde com o aumento de uso ocorre o problema.
- Quando a aplicação se junta a um grupo ou deixa o grupo, existe a chance que a implementação da camada *multicast* confiável (IGRM) acuse *ConcurrentModificationException.*
- Quando se envia dados com Group.send() os dados não podem ser maior que 64k. Restrição do serviço UDP no JAVA.
- O grupo faz partição e junção repetidamente de forma desnecessária durante o tempo de vida da aplicação. Devido a problemas de performance, o coletor de lixo faz algumas coletas de forma demorada as vezes e assim algumas theads são suspensas e nenhuma mensagem é processada neste período dando a falsa impressão de problema na rede.

Como conclusão, devido a estes diversos problemas, uma certa instabilidade do software e uma performance que deixa a desejar neste ambiente JAVA, ve-se que no futuro pode ser um bom mecanismo de comunicação em grupo para ser usado em máquinas rápidas e aplicações que tenham afinidade com o ambiente JAVA.

6 JGROUP

O protocolo Jgroup é o resultado de uma tese submetida à Universidade de Bologna, Padova, Veneza, Itália por Montresor (2000), em janeiro de 2000. Baseada no paradigma de orientação a objeto, todo o trabalho foi desenvolvido de forma a se abstrair a complexidade do sistema e promover a reusabilidade e modularidade. Todo o trabalho é baseado nos conceitos de orientação a objeto como abstração, encapsulamento, herança, polimorfismo e permitindo uma interação cliente/servidor entre objetos distribuídos. Esta tese propõem o paradigma do grupo objeto onde funções de serviços distribuídos são replicados formando uma coleção de objetos servidores logicamente relacionados formando em conjunto um grupo objeto. Um grupo constitui uma facilidade de endereçamento lógico: clientes interagem de forma transparente com grupos de objetos invocando remotamente métodos destes objetos, como se fosse um único objeto remoto não replicado.

JGROUP é uma extensão do modelo de objeto distribuído JAVA, baseada no paradigma de comunicação em grupo. Diferentemente de outras extensões orientadas a grupo de modelos existentes de objetos distribuídos, JGROUP é expressamente apontado para suportar o desenvolvimento de aplicações distribuídas confiáveis e com alta disponibilidade em ambientes com particionamento de rede. JGROUP permite a criação de grupos de objetos remotos que cooperam entre si na busca de um objetivo comum usando um serviço de comunicação em grupo que aceita particionamento de rede. Grupos de objetos remotos simulam o comportamento de objetos remotos padrões, implementando um conjunto de interfaces remotas e permitindo a clientes invocar remotamente os métodos definidos nestas interfaces através do mecanismo padrão de RMI do JAVA. JGROUP é complementado por um serviço de estado de intercalação, o qual simplifica o desenvolvimento de protocolos de intercalação, necessários para restabelecer um estado consistente após desaparecer o particionamento.

6.1 Características

O JGROUP é uma extensão do paradigma de grupo de objetos de forma a suportar aplicações cliente do particionamento da rede. Para alcançar estes objetivos, alguns pontos têm que ser considerados:

- Quando se desenvolvem aplicações que se preocupam com o particionamento da rede, os servidores incluídos em uma visão precisam estar cientes se ocorrer um particionamento. Para controle de membros de grupo de uma partição primaria não é conveniente que sejam notificados os serviços sempre que ocorrer uma mudança de cenário devido à falha. De forma contrária todos os servidores participantes de uma computação que se preocupa com partição de rede, devem ser continuamente informados sobre o cenário corrente de particionamento, para se reconfigurar entre eles e ajustar seu comportamento conseqüentemente.
- Em um sistema de grupos de objetos o comportamento de um serviço de comunicação confiável é altamente dependente do serviço de controle dos membros do grupo adotado. Se enfocarmos apenas em partição primária, todas as comunicações são endereçadas para o servidor incluído na visão primária. O projeto das aplicações é simplificado pois temos um único fluxo de comunicação no sistema. Aplicações que se preocupam com particionamento, contudo, podem ter diferentes requisitos. Cada servidor precisa estar apto a comunicar com servidores incluídos em sua visão e clientes têm que poder contactar pelo menos um servidor incluído na sua partição. Cada comunicação envolvendo o grupo precisa seguir as semânticas de sincronismo de visão: dois servidores que instalam o mesmo par de visões consecutivas e na mesma ordem, entregam o mesmo conjunto de eventos de comunicação no tempo ocorrido entre as duas instalações.
- Quando se considera comunicação em grupo que tratam particionamento existe a possibilidade que servidores em partições concorrentes continuem a servir operações externas independentes e gerando inconsistência. Quando as condições que estavam causando o particionamento deixam de existir uma aplicação específica usando protocolo de intercalação de estado tem que ser executada para definir um novo estado global que reconcilia as divergências que estavam ocorrendo. De forma geral, protocolos de intercalação de estado, são mais complexos que o correspondente protocolo de transferência de estado em partições primárias: o anterior é baseado em trocas de informações bidirecionais entre duas partições que estão se intercalando, enquanto o último é

baseado na transferência de informação de servidores já pertencentes a visão primária para servidores que dinamicamente se juntaram a partição primária.

Como têm diferentes requisitos, aplicações que tratam particionamento, não podem ser baseadas no sistema tradicional de grupos de objetos que consideram o enfoque de partição primária. Por esta razão JGROUP provê em contrapartida uma versão dos serviços anteriormente listados para o ambiente particionado:

- Serviço de membros do grupo particionaveis, PGMS (*Partitionable Group Membership Service*) – A tarefa do PGMS é manter os membros continuamente informados sobre o cenário corrente de falhas, incluindo tanto particionamento como queda do servidor. Aplicações que tratam particionamento são programadas de forma a se reconfigurarem por si só e ajustar o comportamento baseado na composição das visões instaladas. Na ausência de particionamento, cada servidor ativo deve instalar a mesma visão e esta visão precisa incluir exatamente estes servidores que estão funcionando normalmente. Se um particionamento ocorrer nosso serviço de membros do grupo particionaveis, garantirá que sobre certas condições, servidores ativos na mesma partição, instalarão visões idênticas e que a sua composição da visão corresponderá à composição da partição.
- Serviço de grupo, GMIS (*Group method Invocation Service*) – O serviço de comunicação incluído no JGROUP é estritamente integrado com o PGMS, e assim oferece um serviço ciente do particionamento. Clientes estão aptos a interagir com servidores que pertencem a sua partição corrente, e servidores incluídos numa visão estão aptos a coordenar seu comportamento usando as facilidades dos serviços de comunicação de grupos. Diferentemente de outros sistemas existentes de grupos de objetos, JGROUP provê somente um serviço de invocação do método grupo que gerência todas as comunicações envolvendo membros do grupo. Em outras palavras não apenas clientes interagem com servidores de grupo através da chamada de métodos remotas neles, mas também interação interna ao grupo são baseadas no mesmo paradigma. Como são caracterizados por diferentes propriedades, denomina-se de chamada de métodos a grupos externos estas executadas pelos clientes e chamadas de métodos a grupos

internos esta executadas pelos servidores. O serviço de chamada multi-chamada/multi-resposta incluído no JGROUP tem muitas vantagens com respeito à comunicação de grupo baseada em mensagens do tipo *multicast.* O desenvolvimento de complexas consistências nos protocolos cientes de particionamento é simplificado já que operações de empacotamento e desempacotamento de parâmetros e disparo de chamadas são manuseadas automaticamente pelo GMIS. Programadores de aplicação podem tirar vantagens do paradigma de orientação a objeto sugerido pelas chamadas internas no JGROUP; em particular mecanismos tais como encapsulamento, herança e polimorfismo podem ser particularmente úteis quando se desenvolvem aplicações complexas. Finalmente o modelo de comunicação unificado permite uma especificação mais compacta com respeito ao sistema de grupo de objetos que misturam mensagens *multicast* e chamada de método remoto.

- Serviço de intercalação de estado, SMS (*State Merging Service*) – A tarefa do SMS é suportar aos desenvolvedores no projeto e implementação dos protocolos de intercalação de estado, para suas aplicações. De forma similar à transferência de estado usado em partições primárias, SMS se encarrega de consultar os servidores de forma a obter informações sobre o estado corrente da computação e difundir para outras partições. Membros do grupo precisam prover operações para obter o estado corrente da computação em uma partição e intercalar esta informação com o estado das outras partições.

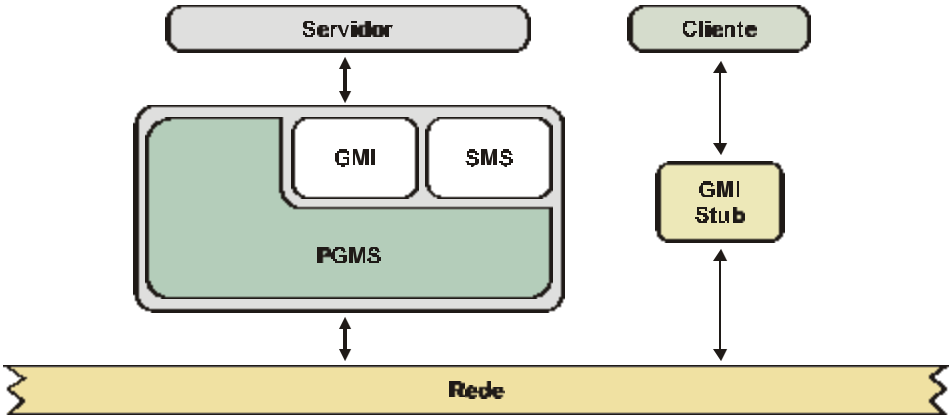

Figura 28 Arquitetura do JGROUP

O PGMS, GMIS e SMS constituem os componentes básicos da arquitetura JGROUP, conforme listado na Figura 28. O objetivo do modelo JGROUP é abranger um conjunto mínimo de requisitos que uma ferramenta para programação orientada a objetos em aplicações que se preocupam com particionamento precisa atender.

6.2 Implementação

Um exemplo de configuração do JGROUP é o que está sendo apresentado na Figura 29. O sistema é composto de uma coleção de máquinas virtuais JAVA conectadas através de uma rede de comunicações. Cada máquina virtual JAVA hospeda uma coleção de objetos. Objetos servidores coordenam suas ações de forma a prover alguns serviços para objetos clientes. Clientes acessam os serviços providos pelos grupos objetos através da invocação de métodos do grupo.

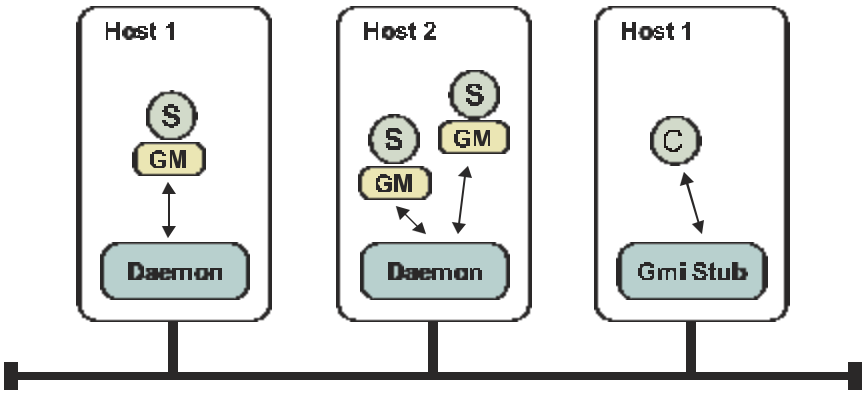

Figura 29 Arquitetura do JGROUP em Máquinas JVM

As facilidades de comunicação em grupo são providas para objetos servidores pelo gerenciador de grupo GM (*Group Manager*) que atua como um *proxy* para executar chamadas internas do método grupo e prove primitivas para o gerenciamento de grupo. Cada servidor é associado com um GM para cada um dos grupos ao qual ele se conecta enquanto cada GM é associado exatamente a um servidor. GM são responsáveis por notificar seus servidores de comunicação de grupo eventos tais como mudanças de visão, além do mais GM também se encarregam de disparar chamadas

internas e externas. Gerenciadores de grupo não implementam todos os serviços de comunicação de grupo descritos anteriormente. Facilidades básicas de comunicação como e controle de membros do grupo são implementadas em um modulo separado chamado JGROUP *Daemon*. Em uma máquina JAVA JVM que está hospedando servidores JGROUP, uma única instância do *daemon* JGROUP é executada. Os *daemons* JGROUP se comunicam entre eles para trocar mensagens e informações de facilidades de acesso.

Existem diversas motivações por trás da escolha de usar este modelo de arquitetura, em contrapartida a ter serviços de comunicação de grupo sendo executados por cada gerenciador de grupo:

- O número de mensagens trocadas para estabelecer comunicação em grupo é reduzido: por exemplo serviços de baixo nível como detecção de falhas são implementados por *daemons* JGROUP um em cada máquina JVM e não um em cada GM.
- Com este modelo é possível distinguir entre objetos servidores locais para uma dada máquina JVM e outros que sejam remotos. Servidores locais para uma dada máquina sempre compartilham o mesmo destino com respeito à falhas: uma falha da máquina JVM causa uma falha em todos os servidores hospedados nela, e se uma máquina JVM se torna particionada do resto do sistema, todos os seus servidores locais param de se comunicar com os servidores remotos. Então dois controles de membros são mantidos de forma distinta, um relativo a servidores e outro relativo a *daemons*. Variações voluntárias de controle dos membros decorrentes do fato de se juntar ou sair do grupo são traduzidas em mensagens únicas requisitando a mudança na lista de servidores hospedados em um *daemon*, sem requerer protocolos complexos de negociação.
- Esta arquitetura baseada em *daemons* pode ser extendida e melhorada pela permissão de objetos servidores em uma máquina JVM poder usar um *daemon* rodando em outra máquina JVM. Isto pode ser interessante para escalabilidade, bem como ser possível estruturar o sistema de forma a ter uma única instância do *daemon* JGROUP em cada LAN contendo objetos servidores. Desta forma é possível compartilhar os custos de manutenção do protocolo de controle dos membros entre os servidores

que compõem a LAN, assim reduzindo o número de mensagens trocadas. Esta extensão contudo ainda não está implementada no protótipo JGROUP.

Clientes não necessitam *daemons* JGROUP e gerenciadores de grupo rodando em suas máquinas virtuais; a interação entre clientes e servidores é manuseada por um componente chamado GMI stub. Cada stub permite clientes executar chamadas externas a métodos em um único grupo; assim diversos stubs podem co-existir na mesma máquina JVM.

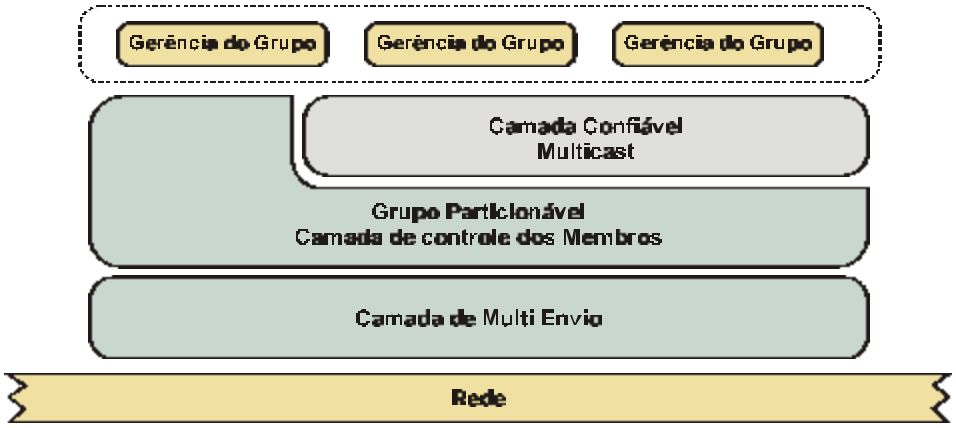

Figura 30 A Arquitetura do *Daemon* JGROUP

Cada *daemon* JGROUP tem uma estrutura em camada como ilustrado na Figura 30, composta dos seguintes componentes lógicos:

- Camada de multi envio, MSL (*Multi-Send Layer*) – Implementar serviços de membro do grupo diretamente sobre serviços de transporte de datagrama sem sequencia, não confiável, ponto a ponto, providos pela rede se torna difícil. A tarefa do MSL é esconder esta complexidade transformando esta primitiva da rede ponto a ponto não confiável, em sua contrapartida melhor-esforço, um para muitos. Informalmente MSL tenta entregar as mensagens enviadas desta forma para todos os *daemons* que estão em uma lista de destinação. Contudo mantém os *daemons* informados sobre qual servidor está acessível e qual está com suspeita de não ser acessível.

- Camada de controle de membros do grupo particionavel, PGML (*Partitionable Group Membership Layer*) - Os serviços providos pela MSL são usados pela PGML de forma a construir e instalar visões como definido pela especificação PGMS. A tarefa da PGML é gerenciar o controle de mebros do grupo de múltiplos grupos atendendo requisições de entrar e sair do grupo, vindas dos servidores e traduzir as possíveis suspeitas de inconsistências geradas pelo MSL em visões válidas para o novo contexto.
- Camada de *multicast* confiável, RML (*Reliable Multicast Layer*) – A tarefa do RML é prover um serviço de *multicast* confiável baseado em mensagens. RML é integrado com a camada de controle de membros do grupo e satisfaz a versão baseada em mensagens das propriedades de sincronismo de visão.

Daemons JGROUP são os blocos básicos para a arquitetura JGROUP; eles provêem os componentes fundamentais do serviço de comunicação em grupo, tais como detecção de falhas, controle de membro do grupo e *multicast* confiável. Facilidades adicionais, tais como chamadas a métodos de grupo e intercalação de estados são atendidas pelo gerenciador de grupos. A arquitetura interna do gerenciador de grupos é altamente modular e foi projetada para possibilitar facilmente a inserção de novas camadas implementando facilidades adicionais não descritas neste trabalho. Por exemplo, seria possível inserir camadas de ordenação para executar ordenação invocando método grupo com semânticas do tipo FIFO, causal e atômica.

Figura 31 Dois Possíveis Empilhamentos das Camadas de Gerenciamento de Grupos

Na Figura 31 ilustram-se as configurações que o gerenciador de grupos pode assumir na versão corrente do JGROUP. Três diferentes camadas podem compor o gerenciador de grupos:

- Camada de interação com *daemon*, DIL (*Daemon Interaction Layer*) – A tarefa desta camada é tratar a interação entre gerenciador de grupos e *daemons*. Esta camada transmite primitivas básicas de comunicação em grupo (tais como join, leave e mcast) para o *daemon* JGROUP que está rodando no sistema; além disso, enfileira as chamadas vindas do *daemon* JGROUP e entrega elas para a camada superior.
- Camada de chamada do método grupo, GMIL (*Group Method Invocation Layer*) – GMIL usa os serviços providos por RML através de DIL para prover os serviços de comunicação conforme o paradigma de comunicação em grupo. Como as informações relativas ao sincronismo de visões são tratadas pelo RML, a única tarefa desta camada é gerencias os problemas relacionados aos métodos de disparo, concatenar e desconcatenar parâmetros e coletar valores de código de retorno.
- Camada de intercalação de estado, SML (*State Merging Layer*) – Esta camada provê as facilidades de intercalação de estados.

Para inserir uma camada no gerenciador de grupos, é necessário implementar duas interfaces: Member para interceptar e enviar notificações de comunicação de grupo chamadas pelas baixas camadas e GroupManager para aceitar primitivas de gerenciamento de grupo chamadas pelas altas camadas e enviar então para camadas inferiores. A única exceção é o *daemon* da camada de interação que implementa somente a interface GroupManager e interage diretamente com o *daemon* JGROUP.

6.3 Resultado Prático da Instalação

Este mecanismo de comunicação de grupos, JGROUP, está disponível em sua pagina na Internet para ser feito *download* (MONTRESOR, 2000). São fornecidos tanto

o código fonte de todos as classes, arquivos com terminação java, bem como uma documentação em HTML para todas as classes e seus métodos. Tem *makefiles* para a compilação de todas as classes e exemplos de uso para ilustrar o uso do JGROUP.

A instalação foi feita em duas máquinas, uma com ambiente WINDOWS e outra com ambiente LINUX. O software tem evoluído nos últimos tempos tendo inclusive diversas versões disponíveis e a cada nova versão além de correção de falhas, a documentação para o uso do software tem evoluído bastante.

Um dos exemplos que acompanham mostra o funcionamento de um servidor de grupos e um cliente que quando faz o contato com o servidor, recebe uma hora gerada no servidor. Na Figura 32 que segue apreseta-se um exemplo da tela durante a execução do servidor Hello Server e do cliente.

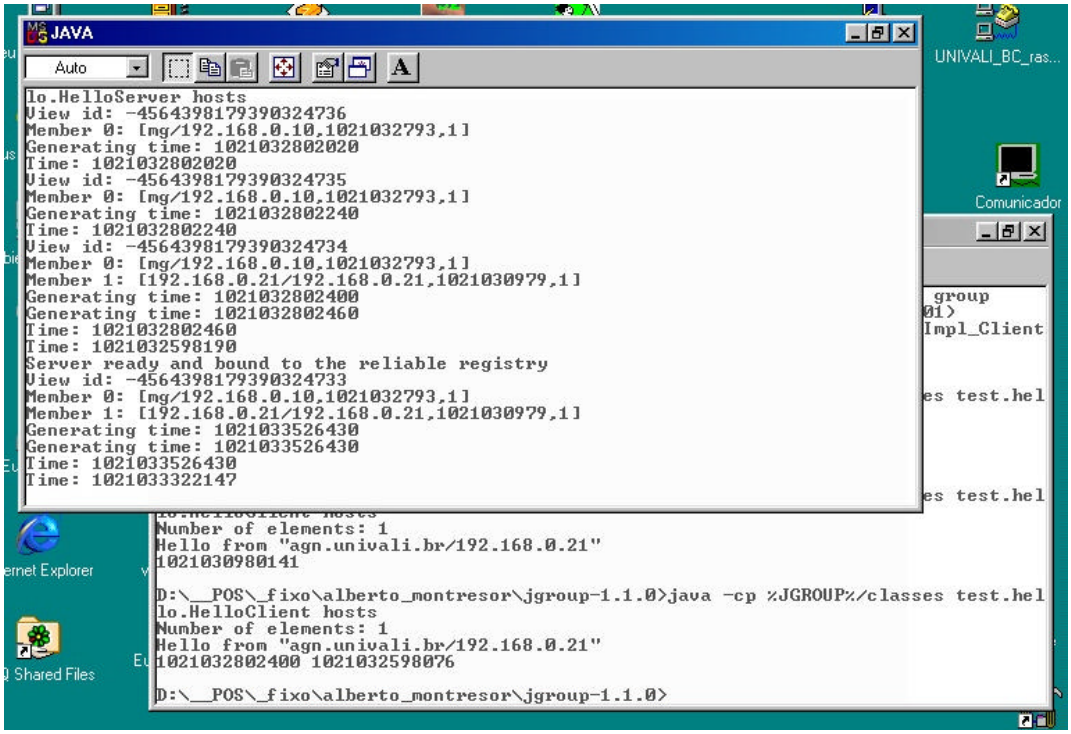

Figura 32 JGROUP Exemplo Hello Server

Pode se observar na tela que está na frente, o programa HelloServer que no inicio tem um único membro do grupo e gera um Time. Em seguida outro membro do grupo é ativado na outra máquina, outro servidor HelloServer, que é mostrado na tela. O primeiro membro do grupo “Member 0” com endereço mg/192. 168.0.10 (WINDOWS) e o segundo membro do grupo “Member 1” com endereço 192.168.0.21 (LINUX). Para

cada membro do grupo que se junta ao grupo, um novo servidor HelloServer, é gerado um Time associado a este servidor.

Para os programas clientes, cada execução vai consultar o HelloServer e receberá o Time do servidor. Assim na mesma Figura 32 na tela que está sobreposta tem-se a execução do programa HelloClient mostrando primeira resposta com um servidor ativo onde recebe o Time daquele servidor e em seguida a resposta com dois servidores ativos mostrando agora dois valores de Time dos dois servidores.

A ativação dos *Daemon* que tratam o JGROUP é feita anteriormente através da execução da classe rregistry. Para todas estas aplicações sempre é necessário informar um arquivo de parâmetros, contendo o endereço IP ou o nome da máquina (Resolução via DNS) onde estão os *Daemon* do JGROUP.

Outra aplicação de teste avalia a performance de chamadas de métodos a grupo externos (GMI). Um programa servidor e um cliente trabalham em conjunto para a realização deste teste de velocidade. Na Figura 33 apresenta-se as telas do programa cliente em primeiro plano e no fundo do Servidor para medida de performance do GMI.

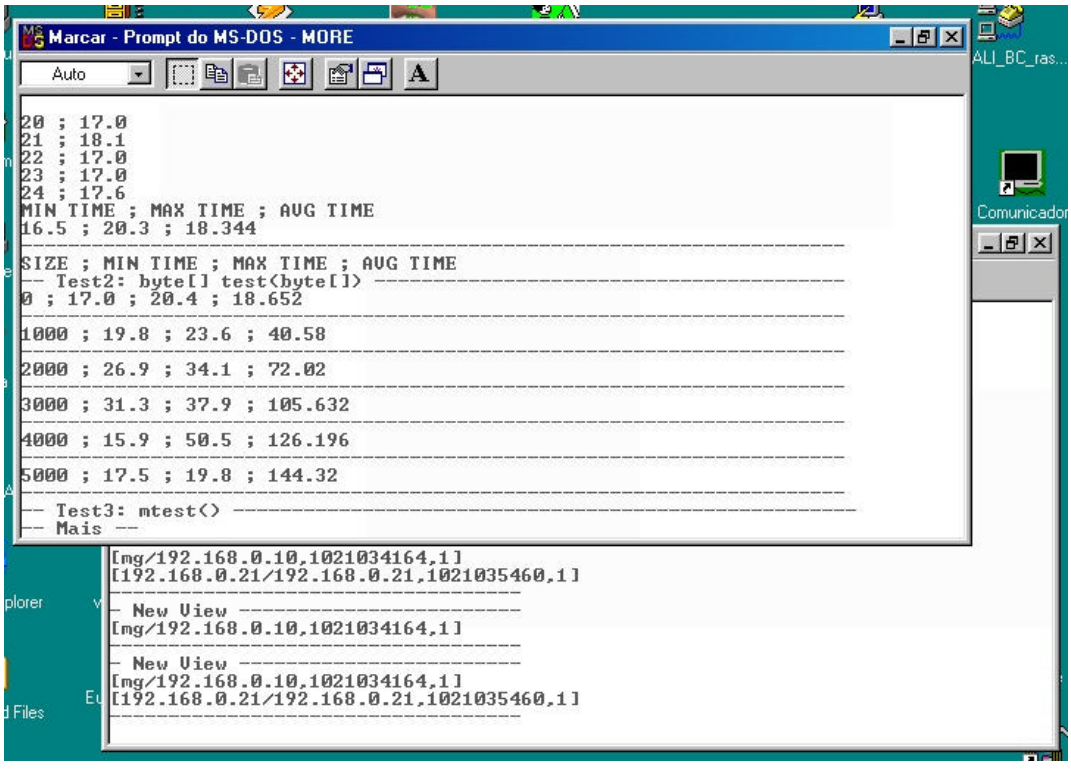

Figura 33 Aplicação de Medida de Performance Servidor GMI

Também um arquivo log é gravado para cada cliente, contendo os resultados dos testes. O teste consiste em diversas chamadas consecutivas de um método, medindo para cada chamada o tempo que leva para ser executada esta chamada. Ao final de 24 chamadas diferentes é apresentado o tempo mínimo, o tempo máximo e o tempo médio das chamadas usando o GMI do JGROUP. Inicialmente são feitos dois conjuntos de testes sendo o primeiro para chamadas *anycast/unicast* sem argumentos passados como parâmetro e sem retorno de valores e o segundo passando como parâmetros um *array* e tendo valor de retorno. Neste segundo teste são 5 chamadas com o tamanho do *array* variando de 1000 a 5000 bytes com incremento de 1000. Para complementar o teste estes mesmos dois conjuntos de procedimentos anteriormente descritos são repetidos agora porém para *multicast.*

Um último teste semelhante ao anteriormente descrito foi realizado e mede a performance usando as mesmas rotinas da medida anterior, porem com chamadas usando o método de invocação remota do JAVA o RMI. Tanto para as chamadas sem parâmetros como para as chamadas passando *arrays*, os tempos foram menores usando JAVA RMI ao invés do GMI do JGROUP.

Como conclusão pode-se dizer que este mecanismo de comunicação em grupo, o JGROUP, é muito interessante por usar o conceito de *Daemon* que pode centralizar o controle em um determinado segmento da rede. Também pode-se ter mais do que um *Daemon* no mesmo segmento para melhor distribuir o controle dos clientes que usam o mecanismo de grupo. Durante os testes práticos simulou-se a queda de rede entre dois *Daemons,* forçando um particionamento e quando se reconectou, a junção foi realizada com sucesso, mostrando uma nova visão do grupo agora com todos os componentes do grupo. Quanto à performance por ser em ambiente JAVA, deixa muito a desejar. Como vantagem tem todos os conceitos modernos da orientação a objetos, permitindo uma perfeita aplicação destes conceitos em um ambiente de comunicação em grupos, orientado a objetos. Cabe ressaltar que o ambiente necessário para testar estas aplicações tem que ser configurado adequadamente, ter máquina JAVA instalada em cada cliente e uma configuração de parâmetros bem detalhada e complexa. Alguns erros foram notados durante a execução e fica a duvida se eram do aplicativo ou do ambiente JAVA. É fundamental um profundo conhecimento do ambiente JAVA para se usar este mecanismo de Comunicação em Grupo.

7 O MECANISMO DE COMUNICAÇÃO EM GRUPO SPREAD

Este mecanismo de Comunicação em Grupo, chamado SPREAD (AMIR, 1998), será detalhadamente descrito neste capítulo. O SPREAD foi o mecanismo escolhido para ser usado na ferramenta de gerenciamento de recursos que ilustra este estudo e esta ferramenta estará descrita no capitulo 8. Assim as características e recursos deste mecanismo de Comunicação em Grupo são aqui apresentados.

7.1 Introdução

Existem algumas dificuldades fundamentais com comunicação em grupo de alta performance, usando redes de longa distância WAN. Estas dificuldades incluem:

- As características (baixas taxas, quantidade de buffer) e performance (latência e largura de banda) variam largamente em diferentes partes da rede.
- As taxas de perdas de pacotes e latência são significativamente maiores e mais variáveis do que em redes locais.
- Não é tão fácil de implementar de forma eficiente, a ordenação e a confiabilidade sobre os mecanismos de *multicast* disponíveis em redes de grande distância como sobre o *multicast* e o *broadcast* em hardware de rede local. Sobretudo porque o mecanismo disponível de *multicast* por melhor esforço para redes de grande distância vem com limitações significantes.

Devido a estas dificuldades, o trabalho tradicional da comunidade de comunicação em grupo, não tem provido soluções adequadas para redes de banda larga mesmo anos após terem sido desenvolvidas boas soluções para rede local. O volume de trabalho hoje voltado para redes de larga escala vem da comunidade de redes, iniciando pelo IP *multicast* melhor esforço e construindo serviços confiáveis e alguma ordenação com uma semântica muito mais fraca do que na comunicação em grupo. Este software SPREAD tem o ponto de vista da comunidade de comunicação em grupo. Implementa contudo, técnicas e facilidades de projeto que não são diferentes das técnicas usadas para prover confiabilidade sobre IP-*Multicast* e leva disseminação e controle de fluxo para redes de banda larga. Assim um sistema de comunicação em grupo tem extensivo e detalhado conhecimento do sistema, os protocolos usados podem ser mais precisos e

produzirem melhor performance do que muitos protocolos de redes genéricos. Esta melhora contudo, vem com um custo. SPREAD faz mais trabalho por nó e não pode crescer demasiado o número de usuários. Contudo ele pode ser disponibilizado a rede de grande distância e pode crescer em número de grupos instalado na rede já que não precisa ser mantido tabela de estado nos roteadores da rede para cada grupo. O sistema de comunicações em grupo SPREAD trata as dificuldades encontradas em redes de grande distância através de três principais características do projeto:

- SPREAD permite o uso de diferentes protocolos de baixo nível para prover disseminação confiável de mensagens, dependendo da configuração da rede. Cada protocolo tem diferentes parâmetros de otimização aplicados a diferentes partes da rede. Em particular SPREAD integra dois protocolos de baixo nível: um para rede local chamado *RING* e um para rede de larga escala chamado *HOP.*
- SPREAD usa a arquitetura tipo cliente servidor, "cliente-*daemon*". Esta arquitetura tem muitos benefícios, o mais importante para ajustes em grande distância é a habilidade de pagar um preço mínimo para as diferentes causas de alterações de membros do grupo. O fato de um processo se associar a um novo grupo ou deixar este grupo se resume a uma simples mensagem. Um processo se desconectar ou conectar ao *daemon* não paga o alto custo envolvido em trocar as tabelas de roteamento na rede de grande distância. Somente as partes que compõem as redes locais necessitam trocar entre si estes roteamentos.
- SPREAD separa o mecanismo de disseminação e segurança local do protocolo global de ordenação e estabilidade. Esta separação permite que mensagens possam ser enviadas a rede imediatamente independente de outros controles de perda e ordenação. Isto também permite selecionar quando mensagens de dados são somente necessárias de serem enviadas a um mínimo conjunto de componentes da rede, sem comprometer a forte garantia semântica que temos em grupos de comunicação típica. Em particular, SPREAD suporta o modelo de sincronismo virtual estendido (MOSER, 1994).

SPREAD tem facilidades na configuração, permitindo ao usuário o controle sobre o tipo do mecanismo de comunicação e o layout da rede virtual. SPREAD suporta canais prioritários para a aplicação. Canais prioritários aceleram o envio de mensagens

usando este recurso, enquanto preservam as garantias de ordenação requeridas. Esta característica deve ter sido o primeiro serviço de prioridade implementado em um sistema de comunicação em grupo. Finalmente, SPREAD suporta semântica grupo-aberto onde um processo que envia não precisa ser membro do grupo para enviar uma mensagem a todos do grupo. O ambiente SPREAD suporta aplicações multiplataforma pois está disponível para diversos ambientes UNIX bem como para ambiente WINDOWS e JAVA.

7.2 Arquitetura do Sistema SPREAD

O sistema SPREAD é baseado no modelo *daemon*-cliente onde geralmente *daemons*, que são processos no UNIX que ficam continuamente executando, estabelecem a rede básica de disseminação de mensagens e fornecem serviços básicos de ordenação e controle dos membros do grupo. Aplicações, que chamam rotinas de uma pequena biblioteca para clientes, podem residir em qualquer lugar da rede e irão se conectar ao *daemon* que estiver mais perto para ganhar acesso ao serviço de comunicação em grupo. O uso deste modelo, como oposto de ter os serviços de ordenação e controle dos membros do grupo atendidos por cada aplicação cliente, é muito importante em uma rede de grande distância por que os deamons minimizam o número de trocas no controle dos membros que o sistema tem que fazer através dos pontos de interligação da rede de grande distância. Além disso, fornece uma infraestrutura básica e estável na qual se pode construir um eficiente controle de roteamento e ordenação para redes de grande distância. O inconveniente que se tem ao usar um modelo de dois níveis *daemon*-cliente é que pode ocorrer interferência entre dois diferentes clientes que usam a mesma configuração do *daemon*. Isto é amenizado pela possibilidade de se rodarem múltiplas configurações dos *daemon*, cada uma servindo diferentes aplicações. Também, pelo modelo de disseminação usado pelo SPREAD, que somente manda mensagens de dados para aqueles *daemons* que necessitam das mensagens minimizando o custo de atividades extras nos *daemons* que a aplicação não está usando. Entretanto tem o custo adicional de comunicação interprocesso e chaveamento de contexto.

SPREAD é bastante parametrizado, permitindo que o usuário possa configurar o mesmo de acordo com as suas necessidades. SPREAD pode ser configurado para usar

apenas um *daemon* no mundo todo ou para usar um *daemon* em cada máquina rodando aplicação de comunicação em grupo. A melhor performance, quando não se tem falha na rede, é encontrada se temos um *daemon* em cada máquina. Entretanto usando alguns poucos *daemons* diminui o custo de recuperação. Em principio, os *daemons* SPREAD podem ser colocados nos roteadores das redes onde, por um custo de maior uso de memória (*buffer*) no roteador, a segurança de mensagens importantes pode ser aumentada.

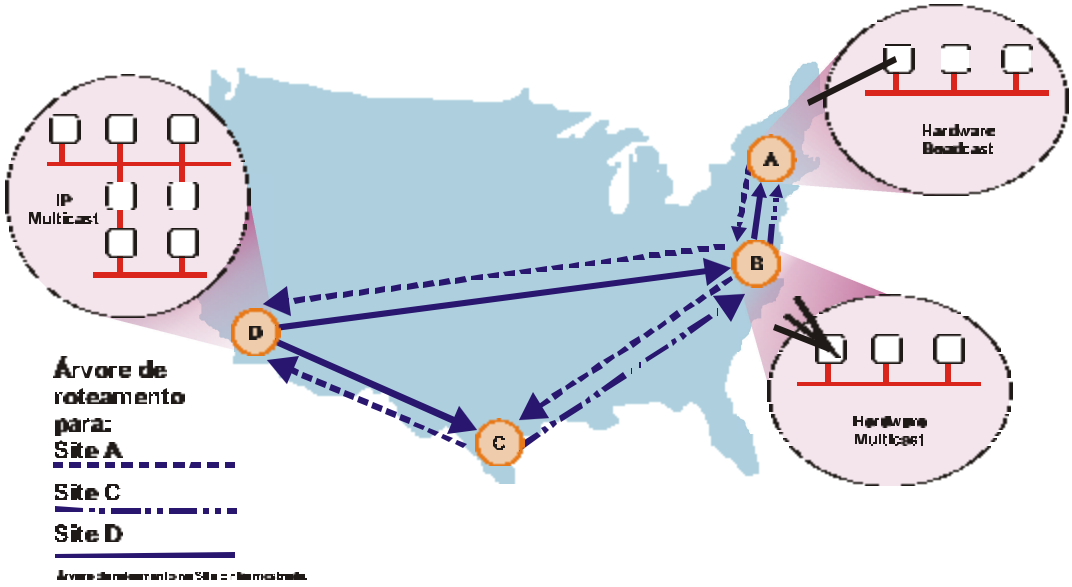

Figura 34 A Configuração de uma Rede de Longa Distância com SPREAD

Um exemplo de rede é mostrado na Figura 34, onde diversos sites são mostrados, geograficamente dispersos, com diferentes custos de link entre eles. Define-se como site uma coleção de máquinas que potencialmente pode encontrar outra máquina através de uma única mensagem, exemplo *broadcast*, hardware *multicast* ou IP *multicast.* Cada site pode ter até dezenas de máquinas nele, o qual não impacta a escalabilidade do sistema SPREAD, já que todas as operações não locais a este site escalam com o número de sites e não com o número total de máquinas envolvidas. Todos os *daemons* participantes de uma configuração SPREAD sabem o completo potencial dos membros participante quando iniciado, mas todo o conhecimento do atual membro participante do *daemon* ativo é obtido dinamicamente durante a operação. Cada site tem um

daemon que atua como representante do site, participando da disseminação na rede de longa distância. Este representante é determinado baseado nos membros participantes neste momento no site local e não é uma configuração fixa por hardware.

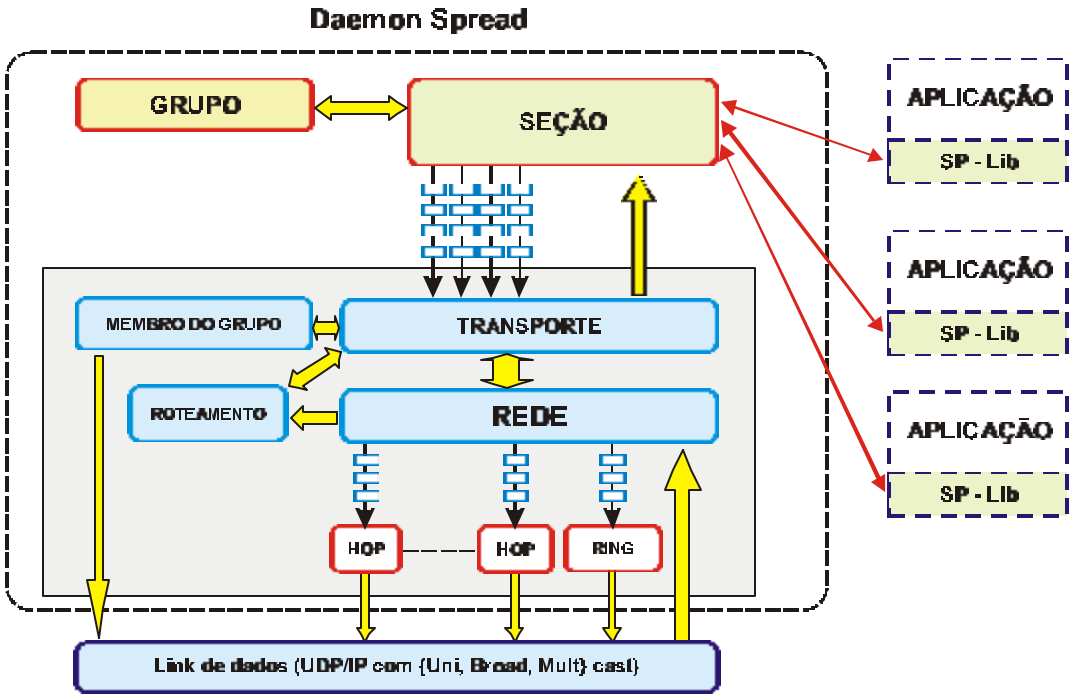

Figura 35 A Arquitetura do SPREAD

A arquitetura do SPREAD é apresentada na Figura 35. Aplicações do usuário são ligadas com uma biblioteca SP_lib (ou usam classes JAVA) a qual prove o completo interface para o cliente , como descrito a seguir. A conexão entre a SP_lib e o *daemon* é uma conexão confiável ponto a ponto, seja IPC ou através da rede. Os módulos de Sessão e Grupos gerenciam as conexões de usuários, gerenciam o processo de participação dos membros no grupo, e traduzem as mudanças de membros do grupo no *daemon* em mudanças de membros do grupo no processo Grupo.

A área sombreada na Figura 35 representa o protocolo interno do *daemon*. Os detalhes destes protocolos serão vistos na seqüência. Alguns principais pontos são:

- Múltiplas filas existem entre os módulos de Sessão e Transporte, um para cada sessão. Isto permite o manuseio de prioridades.
- O módulo de roteamento calcula as árvores de roteamento baseada nos dados do modulo de rede. O módulo de transporte consulta o módulo de

roteamento para determinar os *links* (*HOPs* e *RINGs*) no qual cada mensagem será enviada.

- Diversas instâncias do módulo *HOP* podem existir, cada uma delas representando uma facilidade que é usado por uma ou mais árvores de roteamento.
- No mínimo um modulo *RING* trata da disseminação e confiabilidade no site local se mais de um *daemon* está ativo neste site.
- As instâncias de *HOP* e *RING* podem ser destruídas ou criadas de acordo com as mudanças dos membros do grupo.

SPREAD suporta o modelo de Sincronismo Virtual Extendido EVS (MOSER, 1994), de membros participantes do grupo. EVS pode manusear partições da rede e re-agrupar bem como agregar ou desagregar. Isto provê diversos tipos de confiabilidade (confiável, não confiável), ordenação (sem ordem, FIFO, causal, acordada) e serviços estáveis (seguro) para as mensagens da aplicação. Todos os ordenamentos globais e estabilidade (causal, acordada e segura) são providos através de todos os grupos. Se duas mensagens são enviadas por diferentes clientes para diferentes grupos, qualquer um que tenha se juntado a ambos os grupos recebera as duas mensagens na ordem garantida, mesmo pensando que elas serão recebidas em diferentes grupos. O ordenamento FIFO é provido com cada conexão em um *daemon*, atuando como uma fonte FIFO para propósitos de ordenamento. Assim como no ordenamento global de mensagens o ordenamento FIFO é preservado através do grupo. Em redes de longa distância a entrega confiável se torna útil por que em principio não terá uma penalidade de latência comparado com entrega não confiável por que pode ser entregue tão logo seja recebido. A entrega usando FIFO se torna útil por que uma mensagem somente será bloqueada de ser entregue se uma mensagem anterior a esta da mesma aplicação conectada estiver faltando.

A complexidade no SPREAD está escondida atrás de uma simples mas completa interface de programação de aplicação (API) que pode ser usada tanto para serviços orientados a redes locais como para redes de grande distância sem mudança na aplicação e que prove um modelo claro de comunicação em grupo. Esta API do SPREAD é vista na Figura 36. Uma aplicação pode ser escrita usando apenas 5 funções (SP_connect, Sp_join, SP_leave, SP_multicast, SP_receive) enquanto a interface de programação da aplicação completa permite funções avançadas tais como envios e recebimentos espalhados (*scatter-gather*), envio multi grupo, *polling* em uma conexão

ou comparação de identificação de grupos. O detalhamento desta API usada no SPREAD está descrito adiante em 7.4.

```
SP_connect( char *spread_name, char *private_name, int priority, int group_membership
            mailbox *mbox, char *private_group )

SP_disconnect( mailbox mbox )

SP_join( mailbox mbox, char *group )

SP_leave( mailbox mbox, char *group )

SP_multicast( mailbox mbox, service service_type,
              char *group,
              int16 mess_type, int mess_len, char *mess )

SP_receive( mailbox mbox, service *service_type, char sender[MAX_GROUP_NAME],
            int max_groups, int *num_groups, char groups[][MAX_GROUP_NAME],
            int16 *mess_type, int *endian_mismatch,
            int max_mess_len, char *mess )

SP_error( int error )
```

Figura 36 A Interface de Programação de Aplicação API - SPREAD

7.3 Protocolos

O SPREAD trata com protocolos específicos para ligação local entre os *daemons* ou para ligação via rede WAN entre os *daemons*. Detalhes destes protocolos são vistos na seqüência .

7.3.1 Visão Geral

O que se tem de mais importante no sistema SPREAD são os protocolos que suportam disseminação, confiabilidade, ordenação e estabilidade. Duas camadas de protocolos separados atendem este serviço no SPREAD:

- **Camada de rede** – composta de dois componentes:
  - Para o Nível de Link que provê confiabilidade e controle de fluxo para os pacotes, SPREAD implementa dois protocolos, o protocolo *HOP* para conexões ponto a ponto e o protocolo *RING* para domínios *multicast.* Cada protocolo é otimizado para o seu domínio específico.
  - Roteamento que constrói a rede de saída dos *HOP*s e *RINGs* baseado nos membros de grupos que estão conectados no deamon no momento e seu conhecimento sobre a rede sobre a qual está

trafegando. A rede construída é implementada através de uma árvore de disseminação diferente e residente em cada site.

- **Camada de transporte** – esta camada trata da entrega de mensagens, ordenação, estabilidade e fluxo de controle global. A camada de transporte opera através de todos os *daemons* ativos no sistema.

Construindo uma árvore de roteamento em cada site é importante por diversas razões. Esta claro que ter uma árvore otimizada em cada site é mais eficiente do que uma árvore compartilhada. Uma vez que SPREAD não foi criado para escalar além de diversas dezenas de sites (cada site contendo até diversas dezenas de *daemons*) o custo de calcular estas árvores manter e disseminar as tabelas é gerenciável. O benefício é enorme, especialmente porque estados e informações de roteamento são mantidos em cada nó final.

É importante lembrar que o *overhead* de construir estas árvores pode ser amortizado devido ao longo tempo de vida do membro do grupo em cada site. Isto está em contraste com muitos protocolos de roteamento *multicast* os quais assumem que a árvore precisa somente ser construída como uma necessidade, desde que mudanças nos membros participantes do grupo são muito comuns e então a árvore tem que ser refeita rapidamente.

Para utilizar a infraestrutura de rede tão eficiente quanto possível, é necessário enviar o máximo de pacotes completos. De forma a utilizar diferentes tamanhos de pacotes e ter a habilidade de empacotar múltiplas mensagens de usuários ou pacotes de controle em um único pacote de redes, todos os protocolos da camada de link atualmente tratam não pacotes mas objetos abstratos que podem variar de tamanho desde 12 bytes até 700 bytes. Cada pacote enviado na rede é preenchido com quantos objetos forem necessários. Para facilitar o entendimento, todos os protocolos descritos a seguir são em termos de pacotes. Na prática, contudo, a confiabilidade e o controle de fluxo no nível de link são feitos por objetos.

7.3.2 Disseminação de Pacotes e Confiabilidade

O serviço mais básico de qualquer protocolo de comunicação *multicast* compreensivo é a disseminação de mensagens de dados para todos os receptores

interessados em receber as mensagens. Uma mensagem no nível de aplicação enviada a um grupo pode variar em tamanho desde 0 byte até 128 Kb. SPREAD para as mensagens grandes vai fragmentar e remontar de forma que se enquadre no tamanho do pacote da rede básica e para pequenas mensagens vai agrupar até o tamanho do pacote da rede básica, sem introduzir tempo de latência inaceitável.

SPREAD constrói árvores de disseminação com cada site para os membros ativos do grupo formando um nó na árvore, seja folha ou interior. Uma observação chave em roteamento *multicast* de propósito geral é que a estabilidade da árvore de roteamento é muito importante para encontrar roteamentos confiáveis e exeqüíveis. No sistema SPREAD a separação dos membros do grupo em três níveis permite que o roteamento de nível mais alto, seja baseado em uma configuração de sites muito estável, enquanto que aplicações individuais de membros do grupo possam ser altamente dinâmicas. A criação de métricas reais para decidir como se conectar aos sites na mais eficiente árvore está sendo investigada. Atualmente as árvores são construídas aplicando o algoritmo do menor caminho de Dijkstra para o membro corrente do grupo dinâmico, baseado no grafo completo da rede básica com pesos sendo definidos estaticamente.

Assim que as árvores de roteamento são construídas o envio de pacotes de dados baseado nas árvores são feitos segundos os três princípios:

- **Não Bloqueio**: Pacotes são enviados independente da perda de pacotes enviados anteriormente.
- **Retransmissão Rápida**: Os pais imediatos dos *links* onde houve perda tratam as retransmissões dos pacotes perdidos.
- **Corte:** Pacotes não são enviados para *links* filhos da árvore se não tem membro interessado no pacote.

O Não Bloqueio e a Retransmissão Rápida são providos pelo protocolo *HOP* descrito a seguir enquanto o Corte é provido pelo código de procura no roteamento o qual filtra todos os *links* filhos de todos os sites para onde não temos membros de grupos com o pacote destinado. A identificação de qual site está interessado em um pacote é feita quando o pacote é criado no site de origem. A decisão é ligeiramente conservadora, na possibilidade de que algum site não interessado possa receber o pacote, mas cada site que está interessado a este é garantido que vai pegar o pacote. Isto é mais comum durante o período que uma aplicação pede para se juntar a um grupo ou deixar o grupo mas a operação ainda não foi completada. Para prover a garantia de

ordenação e estabilidade, informações de controle e negociação são enviadas a todos os sites.

A disseminação dos pacotes, confiabilidade e controle do fluxo são providas por dois protocolos do nível de link – o protocolo *HOP* para conexões ponto a ponto e o protocolo *RING* para domínios *multicast* e *broadcast*, como descrito a seguir.

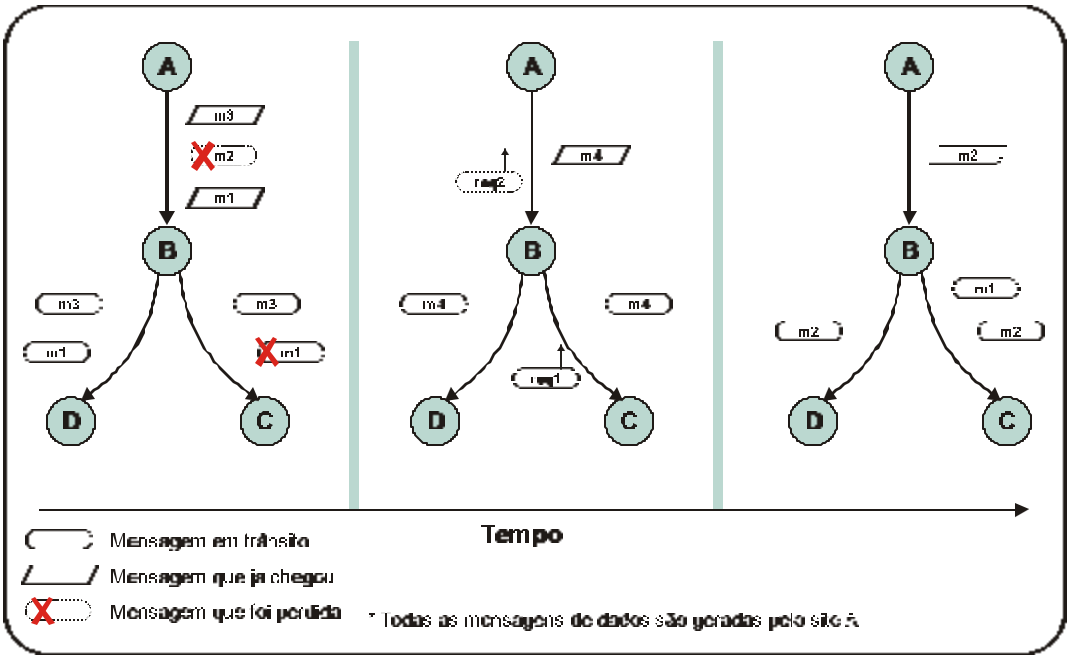

Figura 37 Um Cenário de Demonstração do Protocolo *HOP*

7.3.2.1 Protocolo *HOP*

O protocolo *HOP* opera sobre um serviço de datagramas não confiável como o UDP/IP. A idéia básica do protocolo *HOP* é ter o menor tempo de latência possível quando está transferindo pacotes através de redes passando por diversos pontos de roteamento (*HOPs*) tratando a perda de pacotes em cada trecho *HOP-by-HOP* ao invés de tratar fim a fim. A retransmissão de pacotes acontece imediatamente, entre dois *HOPs*, mesmo que eles não estejam em ordem. Nos casos em que for possível rodar o protocolo SPREAD nos roteadores que conectam sites envolvidos em comunicação de grupos, as vantagens podem ser maiores.

O protocolo *HOP* usa negação de confirmação NACK, para requisitar retransmissão de pacotes e confirmação positiva ACK para determinar até que seqüência de mensagens pode ser removida do buffer de envio. Controle de fluxo é provido por um controle de fichas recipiente furado (*token/leaky bucket*) que limita o número de pacotes que pode ser enviado de uma vez e limita a taxa geral máxima. Em adição, uma janela deslizante, a qual limita o total de pacotes que estão saindo no link, é usada para prevenir arbitrariamente longos tempos de espera para busca de pacotes perdidos. A Figura 37 demonstra o protocolo *HOP*. Um link *HOP* é bi-direcional então cada lado pode enviar para o outro. O lado que envia usa as seguintes variáveis locais:

S_highest_linkseq	O mais alto número de seqüência do link associado a um pacote.
S_other_end_aru	O link de seqüência aru (*all-received-up-to*) reportado pelo outro lado do link.
S_cur_windown	O tamanho corrente da janela deslizante usado para controle de fluxo.

O receptor tem as seguintes variáveis:

R_highest_seq	O maior número de seqüência visto neste link.
R_link_aru	O valor da seqüência até o qual todos os pacotes foram recebidos.
Retransmit list	Uma lista encadeada das seqüências dos valores que foram perdidos e estão aguardando para serem retransmitidos pelo transmissor.

Estas duas últimas variáveis são usadas tanto pelo transmissor como pelo receptor:

Pkt_cnt_linkack	O número de pacotes enviados ou recebidos desde que o ultimo ack do link foi enviado.
Max_pkt_btw_ack	Um valor de ajuste, limitando quantos pacotes podem ser enviados ou recebidos antes de enviar um ack de link.

Três tipos de pacotes são enviados através do link:

- **Dados**: Esta é a porção dos dados da mensagem do usuário ou uma mensagem interna do SPREAD criada por uma camada superior.
- **ACK**: Esta é uma cópia do valor do link_aru corrente do receptor e o highest_linkseq do transmissor.
- **NACK**: Esta é uma lista de valores de todas as seqüências que o receptor perdeu e precisa ser retransmitida.

Tanto os ACKs e NACKs estão limitados ao link no qual eles originam.

Durante uma operação normal o lado que envia em um link *HOP* vai mandar pacotes de dados, ocasionalmente um ACK de link e vai receber ACKs de link vindos do lado do receptor. Isto continua enquanto o receptor está de acordo com o transmissor limitado ao tamanho da janela deslizante e não ocorrem perdas de pacotes.

Quando o receptor recebe um pacote com um número de seqüência maior que seu controle interno (r_hihest_seq + 1), então ele perdeu algum pacote e neste caso adiciona todas as seqüências entre o r_highest_seq e o que recebeu na seqüência do objeto na lista de retransmissão. Neste ponto inicializa um contador para esperar um curto espaço de tempo e ver se os pacotes perdidos estavam apenas atrasados ou fora de ordem na rede, se o tempo expira e ainda tem pacotes na lista de retransmissão então um NACK de link é enviado requisitando estes pacotes. O link de NACK será reenviado com a lista corrente dos pacotes faltantes cada vez que expirar este contador de tempo até que todos os pacotes faltantes sejam recebidos. O receptor mantém um contador de quantas vezes cada seqüência foi requisitada para ser retransmitida. Se este contador assume um determinado valor então o receptor declara o transmissor como morto e inicia uma alteração de membro do grupo. Esta alteração de membro do grupo ocorre mesmo que o receptor tenha recebido outros dados do transmissor e é muito importante para eliminar este problema de falha no recebimento senão o sistema será forçado a ficar bloqueado eventualmente devido ao buffer ficar cheio.

Quando o transmissor recebe um NACK de link ele adiciona os pacotes requisitados na fila de dados a serem enviados e exclui o NACK. Os pacotes retransmitidos serão incluídos como dados enviados no calculo do controle de fluxo. A Figura 38 apresenta o pseudo código para este protocolo *HOP*.

```
Hop_Send_Data(linkid)
     Update token bucket for hop
     while( token is available in bucket for hop )
          Each packet is assigned a unique link_seq value and stored in Link[linkid].open_pkts.
          Send(linkid, pkt)
     if( still_data_available for this hop )
            Event_Queue(Hop_Send_Data, linkid, SendTimeout)

Hop_Send_Ack(linkid)
     ack.link_aru := Link[linkid].r_link_aru
     ack.link_max_sent := Link[linkid].s_highest_linkseq
     Send(linkid, ack)
     Event_Queue(Hop_Send_Ack, linkid, LongHopAckTimeout)

Hop_Send_Nack(linkid)
     add all link_seq values found in retransmit list to nack.req_list[]
     Send(linkid, nack)
     Event_Queue(Hop_Send_Nack, linkid, HopNackTimeout)

Hop_Recv()
     recv(pkt)
     case(pkt.type)
     ACK:
          if (Link[linkid].s_other_end_aru < ack.link_aru)
             remove packets with linkseq < ack.link_aru from Link[linkid].open_pkts[]
             Link[linkid].s_other_end_aru := ack.link_aru
          if (Link[linkid].r_highest_seq < ack.link_max_sent)
             add sequence numbers from r_highest_seq to link_max_sent to retransmit list
             Event_Queue(Hop_Send_Nack, linkid, HopNackTimeout)
     NACK:
          for (i from 0 to nack.num_req-1 )
              req_linkseq := nack.req_list[i]
              pkt := Link[linkid].open_pkts[req_linkseq % MAXPKTS]
              queue_packets_to_send(linkid, pkt)
     DATA_PKT:
          Link[linkid].pkt_count++;
          if (Link[linkid].pkt_count == 1)
              Event_Queue(Hop_Send_Ack, linkid, ShortHopAckTimeout)
          if (Link[linkid].pkt_count > Max_Count_Between_Acks)
              Send_Link_Ack(linkid)
          if (msg_frag.link_seq == Link[linkid].r_highest_seq + 1)
              /* Right on time packet without drops */
              if (Link[linkid].r_link_aru == Link[linkid].r_highest_seq)
                  Link[linkid].r_link_aru++
                  Link[linkid].r_highest_seq++
          else if (msg_frag.link_seq <= Link[linkd].r_highest_seq)
              /* Duplicate or delayed packet */
              remove msg_frag.link_seq from retransmit list and update lowest_missing_linkseq
              if duplicate packet
                  return Null
              if (lowest_missing_linkseq == NONE)
                  Link[linkid].r_link_aru := Link[linkid].r_highest_seq
                  Event_Dequeue(Hop_Send_Nack, linkid)
              else if (msg_frag.link_seq < lowest_missing_linkseq)
                  Link[linkid].r_link_aru := lowest_missing_linkseq - 1
          else
              /* we missed some packets before this one */
              add seq numbers from r_highest_seq to msg_frag.link_seq to retransmit list
              Link[linkid].r_highest_seq := msg_frag.link_seq
              Event_Queue(Hop_Send_Nack, linkid, HopNackTimeout)
          return packet
     Otherwise:
          return packet

* Upper layers will call Event_queue( Hop_Send_Data, Linkid, 0) when data has been queued to send.
```

Figura 38 O Protocolo *HOP*

7.3.2.2 Protocolo *RING*

O protocolo *RING* é usado quando temos mais do que um *daemon* ativo em um site. Note que um site é uma coleção de máquinas que potencialmente alcançam outras

máquinas por uma única mensagem ou seja *broadcast* de hardware, *multicast* de hardware ou IP-*multicast*. Cada membro tem um único identificador que é fixo mesmo ocorrendo quedas e reinícios. O protocolo *RING* é uma modificação do protocolo *RING* usado no projeto TOTEM e TRANSIS (AMIR, 1995). Nestes projetos o protocolo *RING* é usado para prover confiabilidade, fluxo de controle global e ordenamento global.

SPREAD usa o protocolo *RING* para um propósito principal: confiabilidade em nível de pacote e controle de fluxo com o site local, e um propósito secundário: estabilidade de nível de mensagem entre os membros do anel. O ponto crucial é que a mesma ficha (*token*) é usada para todas estas funções. Na mesma circulada da ficha, o algoritmo de cálculo do *RING* aru atualiza tanto o campo de aru do pacote como da mensagem. Assim não estamos pagando nada em complexidade e latência no protocolo para conseguir informações a nível de mensagem. Em contraste com TOTEM e o TRANSIS , ordenamento global e controle de fluxo são providos pelos protocolos de nível de transporte descritos a seguir. Isto limita o uso do protocolo *RING* para tarefas nas quais é mais efetivo: ordenação, confiabilidade e disseminação em área local.

Em outro extremo, a coleção inteira de *daemons* pode ser configurada como um site conectado por roteamento IP-*multicast*. Esta configuração não vai ter a vantagem de nossa confiabilidade no protocolo de rede de longa distância e pode ter uma performance muito pobre.

A ficha (*token*) que circula contém os seguintes campos:

Type	Regular exceto durante as trocas de membro do grupo.
Link_seq	O mais alto número de seqüência de qualquer pacote confiável enviado no anel.
Link_aru	O número de seqüência indicando até que pacote foi recebido de forma confiável por todos os membros do anel. Usado para controlar quando um link pode descartar qualquer referência local ao pacote.
Flow_control	Um contador do número de pacotes enviados ao anel durante a ultima rotação da ficha, incluindo retransmissões.
Rtr list	Uma lista de todos as seqüências de valores que o dono anterior da ficha está pedindo para ser retransmitida.
Site_seq	O número mais alto de seqüência de qualquer mensagem

	confiável originária neste anel. Esta seqüência é local ao site e combinada com a identificação do site provê um identificador único de cada mensagem enviada no sistema.
Site_lts	O mais alto valor LTS visto por qualquer membro no anel mesmo distante. Isto é usado para prover um ordenamento consistente e casual para mensagens seguras e com concordância.
Site_aru	O número LTS até o qual todos os membros do site tenham recebido todas as mensagens com um valor de LTS menor que este.
Site_aru_modifier	O identificador do site membro que modificou o valor do site_aru pela última vez.

Depois de recebido a ficha o *daemon* vai tratar qualquer retransmissão requisitada pelo dono anterior da ficha, depois processa as mensagens recebidas pelas aplicações clientes, envia os pacotes até o limite imposto pelo controle de fluxo, depois atualiza a ficha com novas informações e envia para o próximo *daemon* no anel. Após mandar a ficha o *daemon* vai tentar entregar qualquer mensagem que tenha para as aplicações clientes. Para cada mensagem processada no sistema, um novo site_seq e site_lts são atribuídos e os contadores são incrementados. Para cada pacote confiável enviado um único link_seq é atribuído e o contador incrementado.

Para atualizar os valores do link_aru e do site_aru na ficha, o *daemon* compara o valor local do aru com aquele da ficha, se o valor local é menor, então o valor da ficha é abaixado para o valor local e o campo site_aru_modifier recebe a identificação do *daemon*. Se o valor local é igual ou maior que o valor da ficha, então o *daemon* altera o valor na ficha somente se na variável site_aru_modifier estiver com a identificação deste *daemon*, ou o site_aru_modifier é zero, indicando que nenhum *daemon* tenha diminuido este valor durante a ultima passagem. Todos os membros do anel podem calcular o mais alto valor que eles localmente podem usar tomando o menor deste recém calculado valor aru da ficha e o valor aru da ficha que veio da rodada anterior na ficha.

O protocolo *RING* prove o controle de fluxo, limitando o número de pacotes que cada membro pode enviar durante cada rotação da ficha. O número de pacotes que podem ser enviados em todo o anel para cada rodada, e o limite de quanto cada membro individual pode enviar a cada rodada são parâmetros de configuração de performance. O *daemon* simplesmente envia o mínimo de seu limite individual e o limite total menos o

valor no campo flow_control que foi enviado na ultima vez. Figura 39 apresenta o pseudo código para o protocolo *RING*.

```
Ring_handle_token(token)
     drop token if it is malformed, a duplicate, or the wrong size.
     if( Link[linkid]->highest_seq < token.link_seq ) Link[linkid].highest_seq := token.link_seq;
     answer any retransmit requests which are on the token
     update SiteLTS and SiteSeq values
     Assign site SiteLTS and SiteSeq values to new application messages
     Calculate flow control for this ring
     Send_Data(linkid)
     update tokens flow control fields
     add any link_seq values I am missing to the token req_list[]
     update Link[linkid].my_aru
     update token.link_aru, token.set_aru, token.link_seq, token.site_seq, token.site_lts,
            token.site_aru, token.site_aru_modifier
     send token to next daemon in the ring
     calculate Link[linkid].ring_aru based on token.link_aru and last_token.link_aru
     discard all packets with link_seq < ring_aru from Link[linkid].open_pkts[] array
     calculate site_aru (Highest_ARU[my site]) based on token.site_aru and last_token.site_aru
     copy token to last_token
     Deliver_Mess()
```

Figura 39 O Protocolo *RING*

7.3.3 Entrega de Mensagens, Ordenação e Estabilidade

A camada de transporte, que é a camada superior do protocolo, provê as garantias requeridas para a entrega, ordenação de mensagens e a semântica de serviços estáveis. A camada de transporte usa a camada da Rede para disseminação , controle de fluxo local e confiabilidade na transmissão dos pacotes. Uma mensagem no SPREAD vai tipicamente passar sobre diversos *links* enquanto está sendo enviada a outros *daemons.* Na ocorrência de falhas o protocolo de transporte em cada *daemon* manterá uma cópia completa das mensagens enviadas ou recebidas até que se torne estabilizada através de todo o sistema, e desta forma possa prover a retransmissão de mensagens que foram perdidas durante as trocas entre os membros do grupo, assim que forem disparados avisos de falha. Os protocolos de recuperação em caso de falhas não estão sendo discutidos neste ponto.

Ordenação e estabilidade são providas por um protocolo nível de transporte simétrico que roda em todos os *daemons* ativos. Usa uma seqüência única de valores identificando cada mensagem que origina do site, o identificador único assinalado a cada site, e um tempo de relógio lógico definido pela relação de Lamports (aconteceu antes), para prover um ordenamento total em todas as mensagens no sistema e para calcular quando a mensagem se tornou estabilizada. Estabilidade é definida como uma

mensagem que este *daemon* sabe que todos os outros *daemons* relevantes tenham recebido e então esta mensagem pode ser removida do sistema.

A cada mensagem são assinalados os seguintes valores antes da disseminação:

- Uma identificação do site
- Uma seqüência no site
- Um tempo tipo Lamport (LTS – *Lamport Time Stamp*)
- Uma seqüência da sessão

Conseqüentemente uma mensagem pode ser completamente ordenada baseada estritamente nas informações contidas na mensagem. O uso de um tempo de relógio lógico e valor de seqüência têm um beneficio substancial de ser completamente descentralizado e de fazer a recuperação de partições e intercalar de maneira consistente com a possibilidade EVS. Sem as mensagens ordenadas na origem, entrega segura e com concordância não podem ser alcançadas através de partições da rede.

Mensagens não confiáveis têm uma seqüência de sessão separada por que elas não precisam do suporte requerido para mensagens confiáveis. Já que elas podem ser descartadas nos não queremos que a sua perda interfira com os protocolos que tem confiabilidade. A razão de que elas tenham um valor de seqüência contudo é para ter certeza que o *daemon* não faça entrega de copias duplicadas da mensagem para a aplicação e para ter uma forma de identificar uma especifica mensagem e então o *daemon* possa recompor esta mensagem a partir de seus pacotes. Se alguma porção de uma mensagem, não confiável, não chegar em um especifico tempo após o primeiro pedaço da mensagem ter chegado, então são descartadas todas as porções que chegaram. Mensagens não confiáveis são ainda enviadas somente quando o *daemon* tem a ficha no anel local por causa de assuntos de controle de fluxo. Mensagens não confiáveis são enviadas assim que as mensagens cheguem completas e não fazem parte do protocolo principal de envio descrito a seguir.

Mensagens confiáveis usam a camada de confiabilidade da rede física. Cada link garante transporte confiável dentro de um tempo limite, independente de falhas no processador ou na rede física. Assim confiabilidade fim a fim é provida no caso onde não temos falhas devido a todos os pacotes seguirem através de todos *HOPs* e *RINGs* para todos os *daemons* que precisam deles. Mensagens confiáveis são enviadas tão logo a mensagem seja recebida completa já que não é prevista nenhuma garantia de ordenação. Desta forma o seu envio nunca é atrasado devido a outras mensagens.

Mensagens FIFO têm a mesma garantia de confiabilidade que mensagens confiáveis. Também temos a garantia que será enviada após todas as mensagens da mesma sessão com menor valor de seqüência de sessão. Para prover um ordenamento FIFO por sessão, SPREAD incorre no custo de uma pequena porção de memória por sessão. Em um sistema ativo com muitas sessões este custo é pequeno comparado com as áreas de buffer de mensagens requeridos e pode ainda ter considerável beneficio para latência da mensagem.

Mensagens acordadas são transmitidas em ordem consistente com ambas ordenações FIFO e tradicional Causal. Esta ordenação é consistente através dos grupos como um todo. Para prover este ordenamento com o de acordo, o *daemon* local envia as mensagens de acordo com a ordem lexicográfica {LTS, site id} dos campos da mensagem. O custo é que para ordenar totalmente uma mensagem o *daemon* precisa ter recebido uma mensagem ou pseudo mensagem de atualização de cada um dos outros sites com valor que indica não ter mensagem com um menor {LTS, site id} que já tenha sido transmitido. SPREAD minimiza este custo por que só requer uma mensagem de cada site, não de cada *daemon*. O único potencial método mais rápido requer um seqüenciador o que introduz um recurso centralizado. Assim sendo um seqüenciador não pode prover EVS em redes particionadas. Mensagens seguras são enviadas em ordem consistente com ordenação acordada. Estabilidade é determinada de maneira hierarquica. Cada site usa o campo Site_aru do anel local para gerar um valor All-Received-Upto para todo o site. Este valor representa o status de estabilidade do site local. Este valor ARU é então propagado para todos os outros sites usando a camada de rede física. O valor mínimo do ARU de todos os sites determina a estabilidade global das mensagens. Mensagens seguras iguais ou abaixo deste valor podem ser enviadas e todas as mensagens enviadas iguais ou abaixo destas podem ser descartadas.

SPREAD usa diversas técnicas para otimizar os cálculos necessários para esta ordenação acima, tais como tabelas hash, listas multi-dimensionais ligadas e caching.

7.4 Interface de Programação de Aplicação (API)

A interface de programação de aplicação (API) usada para o SPREAD é composta de poucas chamadas. A seguir temos um detalhamento da forma como é tratada a

situação de área de memória insuficiente para receber uma mensagem, tipos de dados e chamadas das funções no SPREAD.

7.4.1 Tratamento de Buffer Insuficiente

É um comportamento tradicional nas (API) interfaces de programação para aplicações em redes que quando um usuário aloca uma área de buffer insuficiente, a API fornece a maior quantidade de dados que for possível e trunca o resto. Algumas vezes o usuário recebe a noticia que algum dado foi truncado e outras vezes nenhuma notificação é dada. Assim é responsabilidade do usuário detectar quando um datagrama é muito pequeno e recuperar de alguma forma (tal como requisitando novamente o dado).

A dificuldade com o uso deste enfoque no SPREAD é que quando a aplicação tem que recuperar este erro, algumas propriedades da mensagem são perdidas. Por exemplo se a mensagem for do tipo SAFE, os outros membros podem certamente assumir que ou todos os membros recebem as mensagens ou eles não recebem devido a alguma interrupção no processo ou desconexão do SPREAD. Neste caso alguns membros podem pegar parte dos dados , mas tem que recuperar o resto, também os dados podem ser perdidos mesmo quando o processo continua a executar corretamente o que torna difícil para outros detectar a falha.

Essencialmente porque cada mensagem tem significados intrínsecos tais como ordenação, ou garantia de confiabilidade, perdas de dados imprevisíveis por outro lado comprometem em um sistema confiável a semântica valida que queremos usar. É possível detectar esta perda e recuperar, mas os custos são significativos, especialmente quando comparados contra os custos de evitar o problema em primeiro lugar. Assim, diferentemente de datagramas UDP, as mensagens no SPREAD são projetadas para serem confiáveis mesmo quando ocorrer buffer insuficiente.

O método usado é bem definido. SPREAD nunca ira truncar mensagens grandes a menos que explicitamente seja solicitado. Quando você chama SP_receive com um buffer para dados ou para a lista de grupos insuficiente para os dados a serem recebidos, a função SP_receive ira retornar com um código de erro de GROUPS_TOO_SHORT ou BUFFER_TOO_SHORT e nenhum dado ou lista de grupos será retornado. As únicas informações que irão retornar são os seguintes parâmetros:

- Service_type que vai conter o tipo de mensagem.

- Sender — está vazio.
- Num_groups — retornara o número de grupos que o parâmetro groups necessita para evitar que ocorra o erro GROUPS_TOO_SHORT. Este número retorna como um número negativo. Se o parâmetro groups for suficiente o bastante então o campo num_groups será 0.
- Groups — vazio.
- Mess_type — retorna com o tipo de mensagem que a aplicação enviou com a mensagem original, este é apenas um inteiro tipo short int (16 bits).
- Endian_mismatch — recebe o tamanho, em bytes, da área de buffer de dados que seria necessário para receber completamente esta mensagem ser dar o erro BUFFER_TOO_SHORT. Este número é retornado como um número negativo.
- Mess — vazio

Assim, quando SP_receive retorna um dos erros *_TOO_SHORT você pode examinar os campos service_type e mess_type para obter informações sobre que tipo de mensagem o SPREAD está tentando enviar para você. Você pode então examinar os campos num_groups e endiam_mismatc h para descobrir qual o tamanho do buffer que é necessário. Você então aumenta o buffer da aplicação e chama SP_receive novamente.

Esta forma de recuperação é segura com aplicações multi-thread porque cada chamada tem sucesso ou falha na sua própria execução e se duas threads pedem o reenvio de uma mesma mensagem, uma vai receber a mensagem e a outra vai receber a mensagem seguinte a esta (o mesmo que aconteceria se não houvesse erro nem retransmissão).

Esta forma de recuperação contudo requer que a aplicação verifique se ocorreu erro quando chamado SP_receive e se um erro *_TOO_SHORT ocorreu então ou aumenta a área de buffer ou chama SP_receive novamente com um flag DROP_REC ligado, como será descrito a seguir. Se o erro for ignorado ou não for corrigido o tamanho do buffer insuficiente, a aplicação irá continuamente chamar SP_receive, em

um loop e nunca vai receber nada de mensagem (apenas código de erro que não está tratando).

Se a aplicação não quer atualmente receber a área de buffer completa seja para dados ou para a lista de grupos, temos a opção de chamar SP_receive com o campo service_type contendo o valor correspondente a flag DROP_RECV. Quando isto é feito, SPREAD ira tratar a mensagem apenas como a maioria dos sistemas de rede e retorna o total de dados ou lista do grupo que cabe na área de buffer, truncando o restante. Mesmo assim ainda ira retornar um erro informando a aplicação que ela está perdendo dados. Em aplicações simples ou aquelas mais relaxadas, ou com requisitos especiais este tipo de situação, truncar a mensagem, pode ser mais interessante do que realizar os procedimentos completos de recuperação.

7.4.2 Tipos de dados da API

A API do SPREAD usa somente alguns tipos de dados.

```
#define         mailbox         int
#define         service         int
#define         MAX_SCATTER_ELEMENTS   100
typedef         struct dummy_scat_element{
                char  *buf;
                int  len;
                } scat_elemet;
typedef         struct dummy_scatter {
                int  num_elements;
                scat_element  elements
                [MAX_SCATTER_ELEMENTS];
                } scatter;
typedef         struct dummy_group_id {
                int32     id [3];
                } group_id;
```

7.4.3 SP_Funções

A seguir é apresentada a sintaxe de chamada das funções em ambiente de programação usando a linguagem C.

7.4.3.1 SP_connect

```
#include <sp..h>
int  SP_connect ( const char *spread_name, const char *private_name, int priority, int
                  group_membership, mailbox *mbox, char *private_group);
```

SP_connect é a chamada inicial que uma aplicação precisa fazer para estabelecer a comunicação com o *daemon* SPREAD. Todas as outras chamadas posteriores precisam se referenciar a um valor de mbox que será inicializado na chamada desta função (mbox é passado por referência).

O campo spread_name é o nome do *daemon* SPREAD aonde irá se conectar. Será uma string de uma das seguintes formas:

- 4803 — Conecta ao *daemon* SPREAD na máquina local usando UNIX Sockets com socket em /tmp/4803. Não pode ser usado em WINDOWS.
- 4803@localhost — Conecta ao *daemon* SPREAD na porta 4803 da máquina local através do loopback TCP/IP. Esta forma pode ser usada em WINDOWS.
- 4803@host.dominio.br — Conecta a uma máquina identificada pelo nome de domínio “host.domínio.br” na porta 4803.
- 4803@x.y.z.123 — Conecta em uma máquina identificada pelo endereço IP “x.y.z.123” na porta 4803.

O campo private_name é o nome como esta conexão gostaria de ser conhecida. Precisa ser único na máquina que está executando o *daemon*. O nome pode ser de tamanho no máximo igual a MAX_PRIVATE_NAME com caracteres validos para nome de grupo não contendo o caracter “#”.

O campo priority pode ser um flag com 0 ou 1, indicando se esta conexão será uma conexão prioritária ou não. Atualmente não tem efeito.

O campo group_membership é um valor lógico, representado por um inteiro. Se for igual a 1 então a aplicação recebera todas as mensagens relativas a alterações nos membros do grupo para esta conexão, se for 0 então a aplicação receberá apenas mensagens que contém os dados e nenhuma mensagem informando alterações nos membros do grupo, tais como particionamento da rede, a chegada de um membro no grupo, a saída de um membro do grupo, etc.

O campo mbox é um ponteiro para a variável mailbox. Após a chamada de SP_connect no retorno esta variável conterá o valor de mbox para esta conexão.

O campo private_group precisa ser um ponteiro para um string grande o suficiente para comportar pelo menos o número de caracteres igual a MAX_GROUP_NAME. Após a chamada do SP_connect irá conter um nome de grupo privativo desta conexão. Este nome de grupo pode ser usado para envias mensagens do tipo unicast para esta conexão e ninguém poderá se juntar a este grupo especial.

Para a chamada de SP_connect têm-se os seguintes valores de retorno:

- ACCEPT_SESSION — em caso de sucesso na chamada.
- ILLEGAL_SPREAD — o nome dado para a conexão foi ilegal. Usualmente porque foi um *socket* Unix para WINDOWS, um formato impróprio para host ou uma porta ilegal.
- COULD_NOT_CONNECT — chamada de baixo nivel ao *socket* falhou não permitindo a conexão ao *daemon* SPREAD neste instante.
- CONNECTION_CLOSED — durante a comunicação para estabelecer a conexão ocorreram erros e a inicialização não pode ser completada.
- REJECT_VERSION — o *daemon* e as bibliotecas usadas na aplicação têm versões incompatíveis.
- REJECT_NO_NAME — não foi fornecido privat_name.
- REJECT_ILLEGAL_NAMES — o nome fornecido não está de acordo com as regras (tamanho ou caracteres ilegais)
- REJECT_NOT_UNIQUE — o nome fornecido não é único neste *daemon*.

7.4.3.2 SP_disconnect

```
#include <sp..h>
int  SP_disconnect ( mailbox mbox);
```

SP_disconnect precisa ser chamado quando uma aplicação quer terminar a sua conexão com o *daemon* SPREAD. A aplicação pode ter outras conexões ainda abertas para o *daemon* e pode abrir uma nova conexão após ter desconectado.

O campo mbox precisa ser o correspondente à conexão que se está querendo desconectar.

Para a chamada de SP_disconnect têm-se os seguintes valores de retorno:

- NORMAL — retorna 0 em caso de sucesso.
- ILLEGAL_SESSION — quando a seção mbox informada não é uma conexão válida.

7.4.3.3 SP_join

```
#include <sp..h>
int  SP_join ( mailbox mbox, const char * group);
```

SP_join permite que esta aplicação se junte, faça parte do grupo cujo nome é passado no string group. Se o grupo não existe em nenhum dos *daemons* SPREAD ele será criado senão irá fazer parte do grupo já existente, como mais um membro pertencente a este grupo.

O campo mbox é a conexão a ser usada para se juntar ao grupo, primeiro parâmetro e o *string* group representa o nome do grupo ao qual se está pedindo para fazer parte.

Para a chamada de SP_join temos os seguintes valores de retorno:

- NORMAL — retorna 0 em caso de sucesso.
- ILLEGAL_GROUP — o nome do grupo está incorreto por alguma razão. Usualmente porque tinha tamanho zero o tamanho maior que MAX_GROUP_NAME.

- ILLEGAL_SESSION — quando a seção mbox informada não é uma conexão válida. Usualmente porque não está ativa.
- CONNECTION_CLOSED — durante a comunicação ocorreram erros e o processo de juntar-se ao grupo não pode ser iniciado.

7.4.3.4 SP_leave

```
#include <sp..h>
int  SP_leave ( mailbox mbox, cons t char * group);
```

SP_leave deixa de fazer parte do grupo cujo nome é passado no string group. Se o grupo não existe em nenhum dos *daemons* SPREAD esta operação será ignorada, caso contrário deixara de pertencer a este grupo.

O campo mbox é a conexão a ser usada para deixar o grupo, primeiro parâmetro e o *string* group representa o nome do grupo do qual está se pedindo para deixar de fazer parte.

Para a chamada de SP_leave temos os seguintes valores de retorno:

- NORMAL — retorna 0 em caso de sucesso.
- ILLEGAL_GROUP — o nome do grupo está incorreto por alguma razão. Usualmente porque tinha tamanho zero o tamanho maior que MAX_GROUP_NAME.
- ILLEGAL_SESSION — quando a seção mbox informada não é uma conexão válida. Usualmente porque não está ativa.
- CONNECTION_CLOSED — durante a comunicação ocorreram erros e o processo de juntar-se ao grupo não pode ser iniciado.

7.4.3.5 SP_multicast e familia

#include <sp..h>

int SP_multicast (mailbox mbox, service service_type, const char * group, int16 mess_type, int mess_len, const char * mess);

int SP_scat_multicast (mailbox mbox, service service_type, const char * group, int16 mess_type, const scatter scat_mess);

int SP_multigroup_multicast (mailbox mbox, service service_type, int num_groups, const char groups [ ] [MAX_GROUPS_NAME], int16 mess_type, int mess_len, const char *mess);

int SP_multigroup_scat_multicast(mailbos mbox, service service_type, int num_groups, const char groups [ ] [MAX_GROUP_NAME], int 16 mess_type, const scatter scat_mess);

SP_multicast e suas variantes, todas podem enviar uma mensagem para um ou mais grupos. A mensagem é enviada em uma determinada conexão e é marcada como tendo vinda desta conexão. O tipo de serviço, service_type, é um campo que pode ser preenchido com o tipo de serviço que esta mensagem requer. Os tipos de serviços válidos para mensagens são:

- UNRELIABLE_MESS
- RELIABLE_MESS
- FIFO_MESS
- CAUSAL_MESS
- AGREED_MESS
- SAFE_MESS

Este tipo pode ser combinado bit a bit fazendo um ou lógico com outros flags tal como SELF_DISCARD se desejado. No momento apenas SELF_DISCARD é o único flag adicional.

Se as versões de SP_multicast ou SP_scat_multicast estão sendo usadas então somente para um grupo pode ser enviado. Assim o string group precisa incluir o nome do grupo para onde se quer enviar as mensagens. Se a variante multigroup está sendo usada, então os grupos são especificados pelo inteiro num_groups e a tabela de nomes dos grupos chamada groups contendo todos os grupos para onde a mensagem deve ser enviada. Cada grupo tem um nome de não mais do que MAX_GROUP_NAME

caracteres. Esta tabela deve conter no mínimo tantos nomes de grupos quanto for o inteiro num_groups passado também como parâmetro.

A aplicação que envia a mensagem, vai apenas enviar uma única mensagem e o SPREAD vai entregar esta mensagem a todas as conexões que fazem parte de qualquer um dos grupos listados.

O tipo de mensagem é um inteiro (*short* 16 bits) que pode ser usado pela aplicação de forma arbitraria. A intenção é que ele possa ser usado para identificar diferentes tipos de mensagens de dados que precisam ser identificados sem que tenha que se analisar o corpo da mensagem.

Se as variações sem *scatter* (diversas mensagens agrupadas) são usadas, então um buffer único é passado para a chamada especificando uma mensagem completa para enviar. O campo mess_len da o tamanho da mensagem em bytes. O campo mess é um ponteiro para o buffer que contém a mensagem. Para chamadas usando scatter os dois parâmetros são substituídos por um ponteiro, sact_mess o qual vai apontar para uma estrutura do tipo sactter. Isto permite que mensagens compostas de diversas partes possam ser enviadas de uma única vez.

Para as chamadas de SP_multicast têm-se os seguintes valores de retorno:

- NORMAL — retorna o número de bytes enviado com sucesso.
- ILLEGAL_SESSION — a mbox informada não é valida.
- ILLEGAL_MESSAGE — a mensagem tem uma estrutura incorreta, tipo um estrutura scatter não preenchida corretamente.
- CONNECTION_CLOSED — durante a comunicação para enviar a mensagem erros ocorreram e o envio não pode ser completado.

7.4.3.6 SP_receive e SP_scat_receive

```
#include <sp..h>
int SP_receive ( mailbox mbox, service *service_type, char sender
                 [MAX_GROUP_NAME], int max_groups, int * num_groups, char
```

groups [ ] [MAX_GROUP_NAME], int16 * mess_type, int * endian_mismatch, int max_mess_len, char * mess);

int SP_scat_receive (mailbox mbox, service *service_type, char sender [MAX_GROUP_NAME], int max_groups, int * num_groups, char groups [ ] [MAX_GROUP_NAME], int16 * mess_type, int * endian_mismatch, scatter * scat_mess);

SP_receive é a função geral de recepção para o SPREAD. Ela recebe não somente mensagens com dados mas também mensagens de alterações de estado dos membros do grupo para a conexão corrente. Mensagens para todos os grupos associados a esta conexão vão chegar na mesma caixa postal, assim a chamada de SP_receive vai pegar uma única mensagem de qualquer um dos grupos. Após a recepção completa, um número de campos serão inicializados com os valores indicando meta informações sobre a mensagem (tais como grupo, tipo de mensagem, forma de representação interna, etc).

Esta função é a mais complexa usada no SPREAD porque esta é a única maneira para o sistema retornar informação para a aplicação. O significado de cada campo muda dependendo se a mensagem é de dados ou de controle de membro.

Na chamada da função SP_receive a mesma ficará bloqueada, não retorna a aplicação que chamou, se não tiver mensagem para receber.

O campo mbox diz qual a conexão que vai receber a mensagem. O campo Service_type é um ponteiro para a variável do tipo serviço que informa o tipo de mensagem que está recebendo. Este pode ser REG_MESSAGE ou MEMBERSHIP_MESS informando ser uma mensagem de dados regular ou uma mensagem de controle de membro do grupo.

O resto dos parâmetros difere em significado dependendo do tipo de serviço. Se o service_type for REG_MESSAGE (mensagem de dados) então:

- sender — um ponteiro para um *array* de caracteres de tamanho no mínimo MAX_GROUP_NAME. Estará apontando para o nome do grupo que enviou.
- max_groups — o número maximo de grupos que foi alocado espaço para o *array* de grupos passado na função receive.

- num_groups — um ponteiro para um inteiro o qual conterá o número de grupos no *array* groups.
- groups — um array que tem até max_group nomes de grupos, cada um deles com um *string* de até MAX_GROUP_NAME caracteres. Todos os grupos que estão recebendo esta mensagem serão listados neste *array*, a menos que o *array* seja muito pequeno e foi escolhida a opção DROP_RECV.
- mess_type — contém o tipo de mensagem que a aplicação mandou junto com a mensagem original. É um tipo *short* int (16 bits). Este valor é também corrigido de acordo com a representação de *bytes* para números, antes que a aplicação receba o valor.
- endian_mismatch — é inicializado com verdadeiro (1) se a forma como os números são armazenados na máquina que envia é diferente da máquina que recebe (ordem dos *bytes* para representar números que pode variar de uma arquitetura para outra). De outra forma é incializado com falso (0). Este campo é tratado de forma diferente quando certos erros acontecem. Já foi explicado anteriormente o uso deste campo quando o *buffer* de recepção da mensagem é menor que o necessário.
- mess — o corpo da atual mensagem recebida é colocado neste *buffer*.
- max_mess_len — o tamanho da mensagem, mess, em bytes. Mensagens maiores do que o *buffer* são tratadas de forma especial como já descrito anteriormente.

Se a função SP_scat_receive é usada ao invés de SP_receive então os campos mess e max_mess_len são substituídos por uma única estrutura do tipo scatter. Este conjunto de informações na estrutura, podem ser inicializados para conter quaisquer *buffer* que se queira receber e seus tamanhos. Estes *buffer* precisam ser áreas de memória validas e eles serão preenchidos na chamada de recepção na ordem em que eles forem listados.

Se o tipo de serviço for MEMB_MESSAGE (mensagem de membro do grupo) é especificamente uma mensagem de transição dos membros do grupo, TRANS_MESS então os significados dos parâmetros passam a ser:

- sender — aponta para o nome do grupo onde está ocorrendo a alteração.
- max_groups — não usado.
- num_groups — sempre com 0.
- groups — esta vazio.
- mess_type — contém –1.
- endian_mismatch — retorna com 0.
- mess — vazio.
- max_mess_len — não usado

Assim, na essência, a única informação que se recebe é o campo sender o qual contém o nome do grupo que recebe a mensagem de mudança na transição dos membros do grupo. A importância do TRANS_MEMB_MESS é que ela diz a aplicação que todas as mensagens recebidas após ela e antes de menagens REG_MEMB_MESS para o mesmo grupo são mensagens do tipo limpa, o que permite colocar estas mensagens em estado consistente antes que ocorra a mudança de membros do grupo atual.

Se for uma MEMB_MESSAGE e especificamente uma mensagem regular de transição de grupo, REG_MEMB_MESS, neste caso os parâmetros representam:

- sender — aponta para o nome do grupo onde está ocorrendo a alteração.
- max_groups — o mesmo que na mensagem regular.
- num_groups — retorna com o número de membros no grupo após a alteração.

- groups — contém uma lista ordenada de todos os nomes de grupo privados dos membros de grupo após a mudança.
- mess_type — contém o índice deste processo no *array* dos membros do grupo.
- endian_mismatch — retorna com 0.
- mess — contém o identificador deste grupo que está alterando e uma lista de todos os nomes dos grupos dos processos que vem junto com o seu processo proveniente da antiga composição de grupos para a nova composição de membros do grupo.
- max_mess_len — o mesmo que na mensagem regular.

Na área de dados do *buffer* serão incluídos os seguintes campos de tamanho fixo:

- group_id;.
- Int num_members;
- Char trans_members [] [MAX_GROUP_NAME];

O *array* groups terá num-members nomes de grupos, cada um deles como um *string* tamanho fixo. O conteúdo do *array* groups é dependente do tipo de transição que ocorreu conforme segue:

- CAUSED_BY_JOIN — trans_member contém o nome do grupo que se juntou ao processo.
- CAUSED_BY_LEAVE — trans_member contém o nome do grupo que esta deixando o processo.
- CAUSED_BY_DISCONNECT — trans_members contém o nome do grupo que está desconectando do processo.
- CAUSED_BY_NETWORK — trans_members contém os nomes dos grupos da nova configuração que vieram com o meu processo para formar o novo conjunto de membros .

Se for uma mensagem MEMB_MESSAGE e se não for REG_MEMB_MESS nem TRANS_MEMB_MESS então representa exatamente a situação onde o membro

que esta recebendo esta mensagem está deixando o grupo e esta é a notificação que a saída ocorreu, o que é chamado as vezes de mensagem de saída por si próprio. O teste mais simples para mensagens de auto desconexão do grupo e verificar se CAUSED_BY_LEAVE e REG_MEMB_MESS são falsos, então será uma mensagem de auto desconexão do grupo.

As mensagens TRANS_MEMB_MESS nunca terão um tipo CAUSED_BY_type porque elas somente servem para sinalizar onde entrega de mensagens do tipo SAFE ou do tipo AGREED são garantidas para os membros do grupo antes que ocorra a transição de membros do grupo.

Para outros membros do grupo que este membro apenas deixou o grupo eles vão receber um par de mensagens, sendo TRANS_MEMB_MESS e REG_MEMB_MESS como descrito anteriormente numa mudança de membros do grupo.

Os campos de SP_receive quando ocorre um recebimento de auto desconexão do grupo tem as seguintes características:

- sender — aponta para o nome do grupo onde está ocorrendo a alteração.
- max_groups — o mesmo que na mensagem regular.
- num_groups — retorna com 0.
- groups — será vazio. Isto é porque o processo não é mais parte do grupo.
- mess_type — retorna com 0.
- endian_mismatch — retorna com 0.
- mess — contém o group_id do novo membro do grupo e o nome privado do grupo do membro que está deixando o grupo.

O *buffer* de dados vai incluir os seguintes campos de tamanho fixo:

- group_id;
- Int Num_members;
- Char trans_members [ ] [MAX_GROUP_NAME];

O *array* trans_members conterá um nome do grupo privado que está deixando o processo já que esta mudança de membros do grupo ocorreu devido a CAUSED_BY_LAVE.

Para as chamadas de SP_receive e SP_scat_receive tem-se os seguintes valores de retorno:

- NORMAL — retorna o tamanho da mensagem recebida com sucesso. .
- ILLEGAL_SESSION — a mbox informada não é valida.
- ILLEGAL_MESSAGE — a mensagem tem uma estrutura incorreta, tipo uma estrutura *scatter* não preenchida corretamente.
- CONNECTION_CLOSED — durante a comunicação para receber a mensagem erros ocorreram e o recebimento não pode ser completado.
- BUFFER_TOO_SHORT — o *buffer* da mensagem na aplicação é muito pequeno para receber a mensagem completa.
- GROUPS_TOO_SHORT — a area de *buffer* para receber a lista de grupos ou lista de membros que está sendo recebida.

7.4.3.7 SP_equal_group_ids

```
#include <sp..h>
int  SP_equal_group_ids ( group_id g1, group_id g2);
```

SP_equal_group_ids prove uma forma de comparar dois identificadores de grupo que sejam originados de mensagens membro de grupo. Já que o tipo de dado group-id é considerado transparente para a aplicação o que o programador pode fazer é usar este dado ou comparar com outro usando esta função.

7.4.4 Considerações sobre a API

Como se pode observar na interface de programação da aplicação SPREAD encontra-se as funções necessárias para o uso de mecanismo de comunicação em grupo. Além da função que faz a ligação inicial com o mecanismo de grupo, outras funções

básicas estão disponíveis tais como se associar a um grupo, deixar o grupo, enviar mensagens para o grupo e receber mensagens para o grupo.

Um ponto importante a ser considerado é o fato que a função de recebimento de mensagens é única. Assim tanto mensagens contendo dados, mensagens informando alteração da visão ou mensagens de erro, todas são recebidas pela mesma chamada de função. Existe um parâmetro passado por endereço que é alterado na chamada da função de leitura e nos informa se a mensagem que está sendo recebida é de dados ou de transição de grupo. Se for de transição de grupo e do tipo inclusão de um novo membro do grupo, a aplicação recebe um ponteiro que aponta para um *array* com o nome de todos os membros do grupo. Na hipótese deste *array* ter uma dimensão menor do que o numero de membros do grupo, um valor negativo retorna para informar esta situação e não vai ocorrer erro de sobreposição de memória por falha na alocação.

Para informações em maior volume existe a possibilidade de que se passe uma estrutura composta de um número variável de mensagens a qual poderá ser usado tanto na transmissão quanto na recepção das mensagens.

8 AVALIAÇÃO DE DESEMPENHO DO MECANISMO SPREAD

Após a seleção do mecanismo de comunicação em grupo SPREAD para uma avaliação de desempenho, uma série de testes e estudos foram efetuados com o software de forma a termos uma avaliação, a mais detalhada possível. Inicia-se com a codificação de uma aplicação para gerência de recursos em ambientes distribuídos onde um programa de consulta vai interagir com N clientes, membros de um grupo, recebendo informações de gerência de recursos que estão disponíveis nestes clientes. Esta aplicação está descrita a seguir no item 8.1.

Outra avaliação envolvendo um teste de volume, visa avaliar a estabilidade, escalabilidade e performance em um ambiente de rede WAN. Um programa enviando uma grande quantidade de mensagens a um grupo de dezenas de máquinas, medindo os tempos para cada bateria de testes em que se variava o número, tamanho e tipo de ordenação das mensagens. Os detalhes do ambiente testado e os resultados encontrados estão descritos a seguir no item 8.2.

8.1 Exemplo de Aplicação

Para efetivamente termos uma avaliação do mecanismo de Comunicação em Grupo selecionado neste trabalho, definimos uma aplicação de gerenciamento de recursos em ambientes distribuídos composta de duas funções principais, para serem codificadas usando os recursos do ambiente SPREAD. A primeira função básica é a coleta de dados, para o tratamento exclusivo da consulta de tempo de uso de um recurso da rede, na forma de um arquivo de log (arquivo sequencial gravado em ordem cronológica de ocorrência dos eventos) registrando quando um micro é ligado ou desligado, que será descrita no item 8.1.3. A segunda função básica é um gerenciamento em tempo real interagindo com diversos micros clientes. Neste caso a aplicação é composta basicamente de dois programas, na forma de mestre escravo. Nesta aplicação o programa escravo é tratado como um conjunto de escravos que fazem parte do mesmo grupo. Assim cada micro que estiver sendo gerenciado vai executar uma cópia deste programa cliente, fazendo parte do grupo. Um programa mestre, de consulta, enviará mensagens aos programas clientes que identificarão o tipo de consulta e fornecerão a resposta adequada.

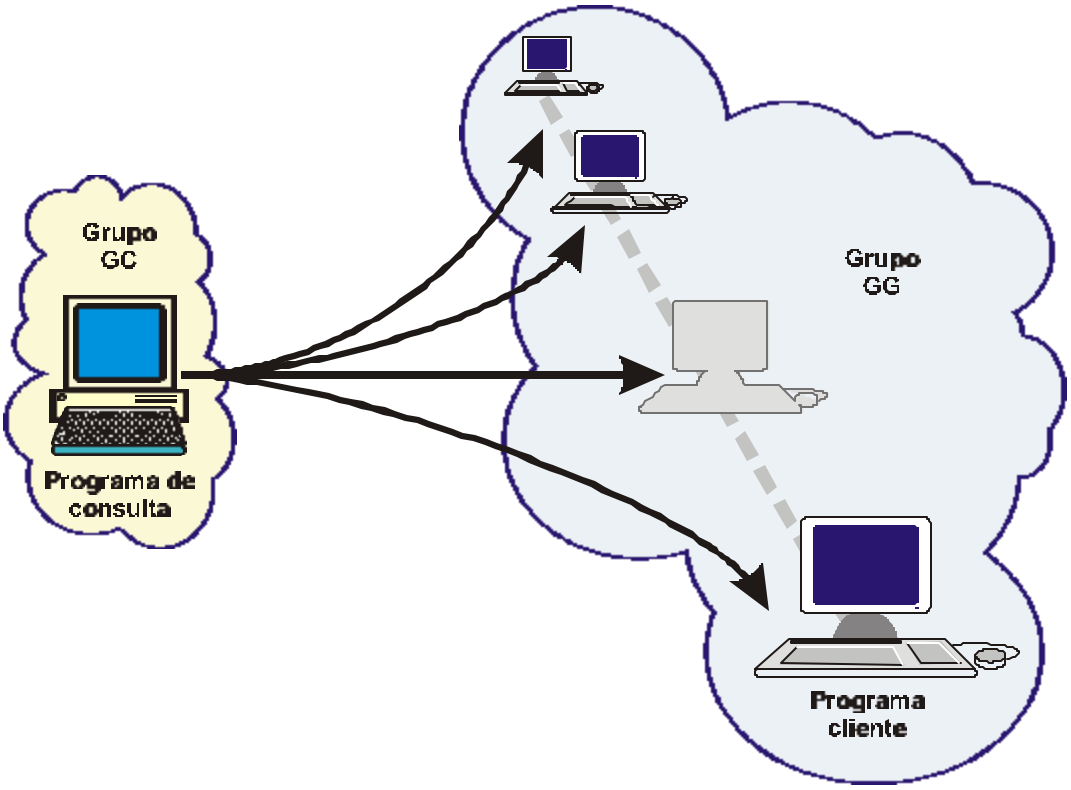

Figura 40 Programa de Consulta e Programas Clientes

8.1.1 Programa Cliente

Este programa foi implementado para trabalhar em ambiente WINDOWS, como um aplicativo (*daemon*) que fica ativo sempre e sem ícone na barra de tarefas. Quando este programa é iniciado, se conecta ao SPREAD com uma identificação única (nome da máquina). Este programa só será desativado via mensagem enviada pelo programa de consulta, mostrado na Figura 40. Todo cliente faz parte de um grupo chamado GG, Grupo Geral. Este cliente é carregado mesmo antes do processo de *login* na estação. Fica sempre aguardando mensagem que vem do grupo GG.

O formato da mensagem é o que segue:

- Código da função 1 byte
- Conteúdo da mensagem variável até 400 bytes

Cód	Conteúdo da mensagem

O programa cliente tem as seguintes funções:

- Função 1 – Mostrar mensagem recebida em uma janela tipo pop-up. Na Figura 41 tem-se um exemplo desta mensagem recebida.

Figura 41 Mensagem do Tipo Pop-Up Enviada a Todos os Clientes

- Função 2 – Informar se o micro (cliente) está livre, sem usuário ativo (sem *login*).
- Função 3 – Verifica se o software XYZ está em uso neste cliente.
- Função 4 – Verifica se existem arquivos com a terminação XYZ neste cliente. Se existir, contar quantos arquivos e o total de k bytes destes arquivos. Responder com o número de arquivos e com o total de k bytes.
- Função 5 – Verifica se um determinado usuário está usando este micro cliente. Confere se o *login* enviado na consulta é o corrente neste cliente e se for responde com a identificação da máquina.
- Função Z – Desativa o cliente. Quando recebe este código Z o cliente encerra a sua execução.

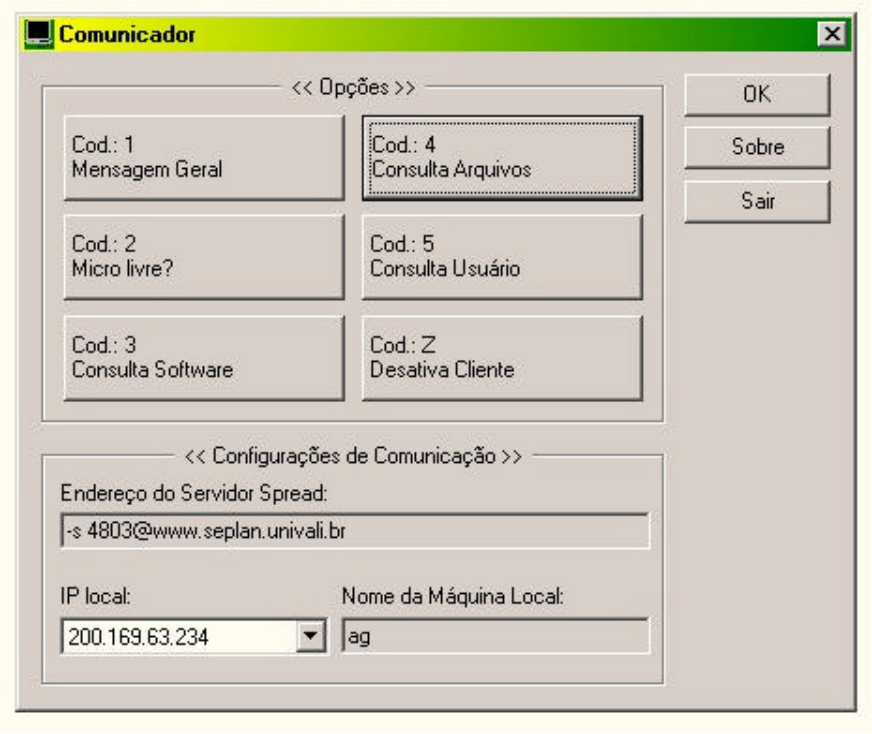

Figura 42 Tela de Comandos do Programa de Consulta

8.1.2 Programa de Consulta

Este é um programa que foi implementado em ambiente WINDOWS que vai interagir com os clientes que estarão fazendo parte do grupo GG. Existe um botão para selecionar cada uma das funções conforme exemplo da tela mostrada na Figura 42. Para cada função selecionada, abre uma nova janela para receber os dados complementares a esta função. Implementa as seguintes funções:

- Função 1 - Mensagem Geral. Esta é uma situação de *broadcast* geral onde uma mensagem é mostrada em todos os clientes. Exemplo na Figura 43. Recebe a identificação do grupo e a mensagem que será enviada a todos deste o grupo.

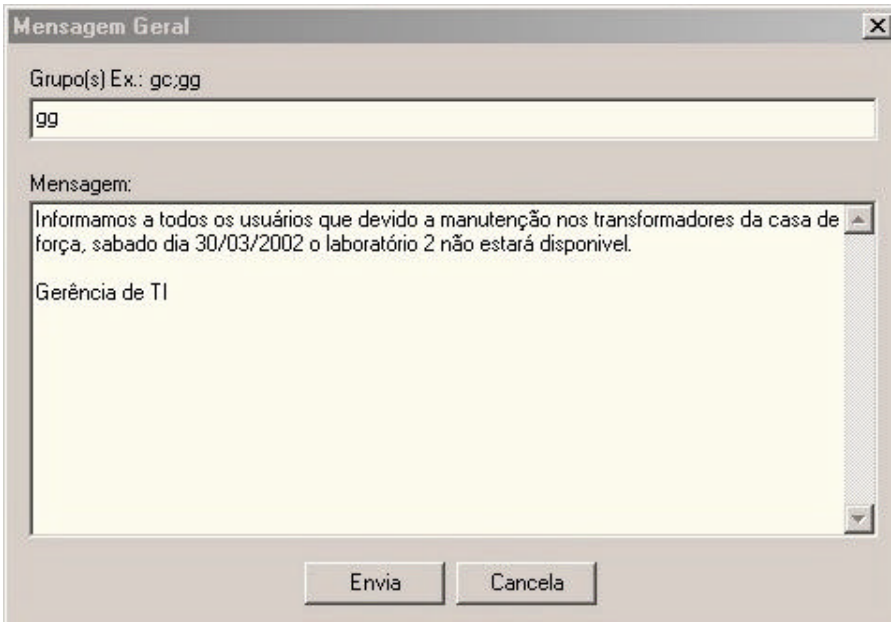

Figura 43 Tela para Envio de Mensagem Genérica

- Função 2 – Consulta de micro livre. Manda uma mensagem de consulta procurando um micro livre e aguarda respostas. Exemplo na Figura 44 Pode receber N respostas que são mostradas em uma janela com barra de rolagem. Existe um campo de entrada correspondente ao tempo de *time-out*, parâmetro na tela já inicializado com 1 minuto, e pode ser alterado pelo usuário.

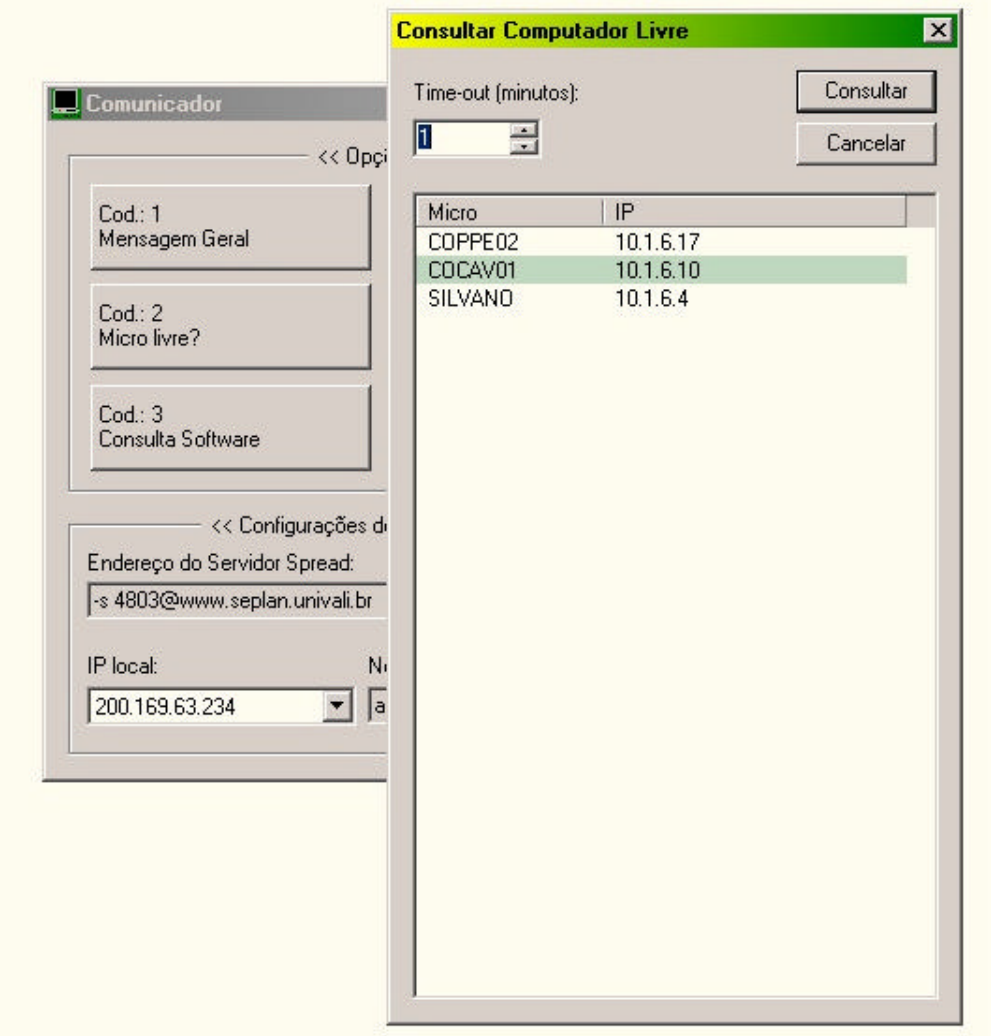

Figura 44 Consulta de Micro Livre

- Função 3 – Consulta de o software XYZ está em uso. Exemplo na Figura 45. Recebe do usuário um campo com o nome do software a ser consultado. Manda a mensagem com código 3, consultando os clientes e aguarda resposta em um determinado tempo como no item 2. Uma janela vai mostrar todas as máquinas que estão executando este software.

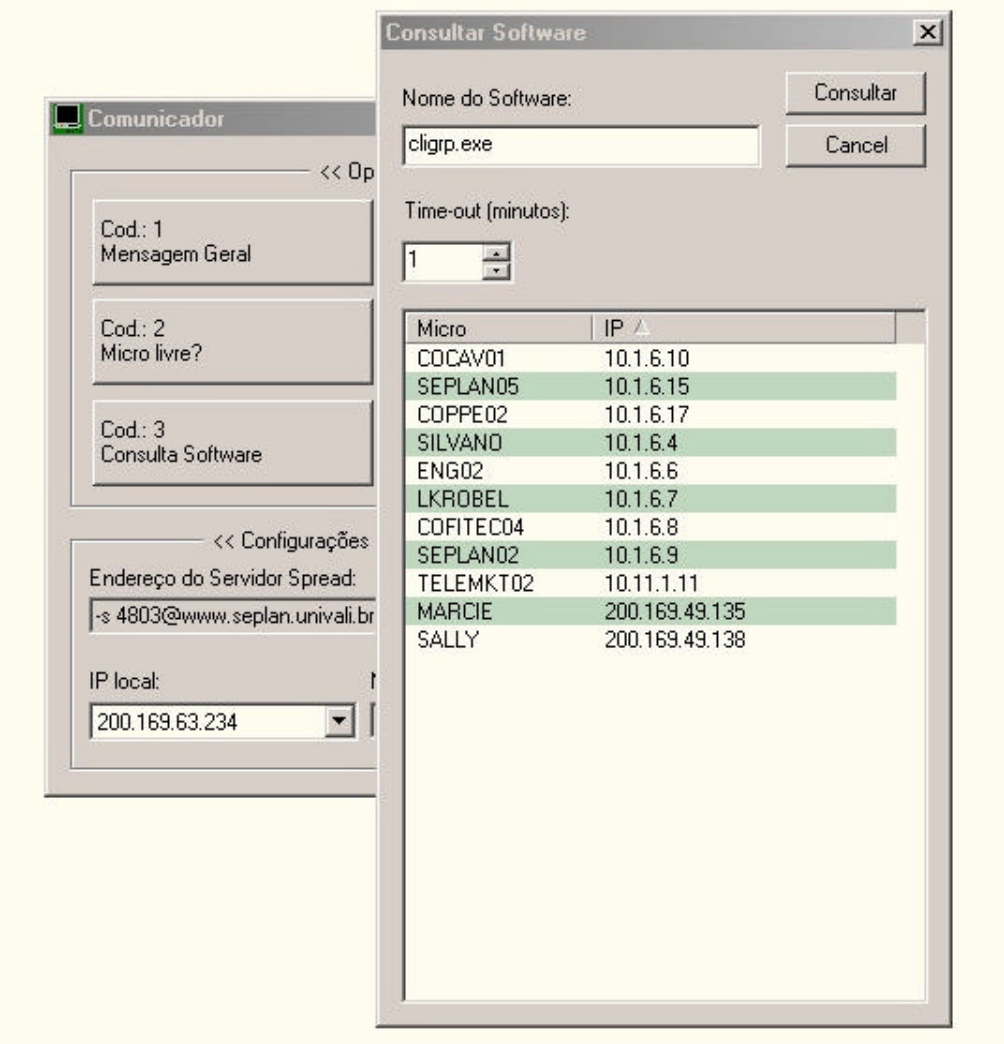

Figura 45 Consulta se Determinado Software Está em Uso

- Função 4 – Consulta se tem arquivos do tipo XYZ na máquina cliente. Exemplo na Figura 46. Recebe um campo com o tipo de arquivo. Manda a mensagem do tipo 4, de consulta a todos os clientes e mostra o resultado em uma janela onde cada linha tem o nome da máquina, número de arquivos e tamanho total dos arquivos. Permite classificar por máquina, número de arquivos, tamanho total. Tem o mesmo controle de *time-out* do item 2.

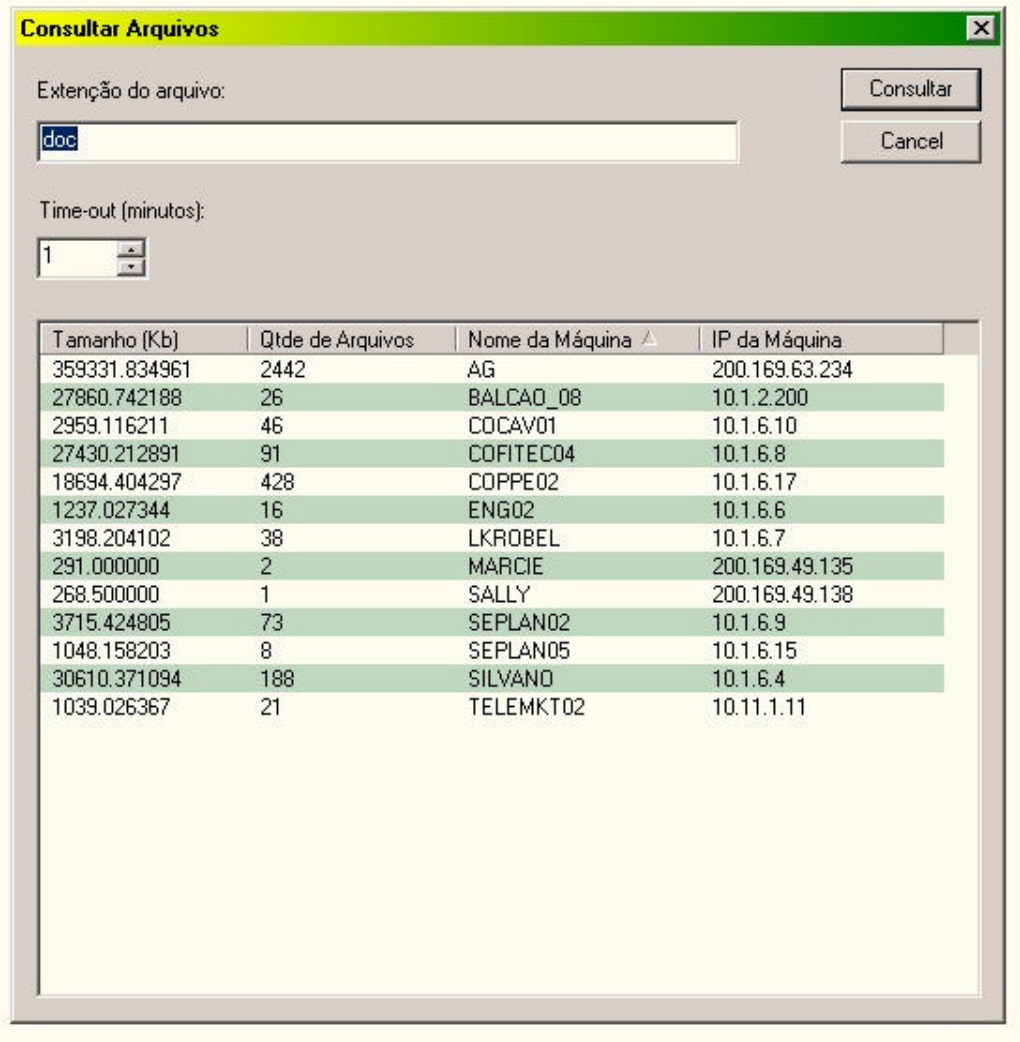

Figura 46 Consulta Tipo de Arquivo em Todos os Micros

- Função 5 – Consulta se um determinado usuário está ativo no micro cliente. Exemplo na Figura 47. Recebe o *login* de um usuário e envia uma mensagem do tipo 5. Se receber resposta informa onde está o usuário, nome da máquina. Usa o mesmo controle de de *time-out* do item 2.

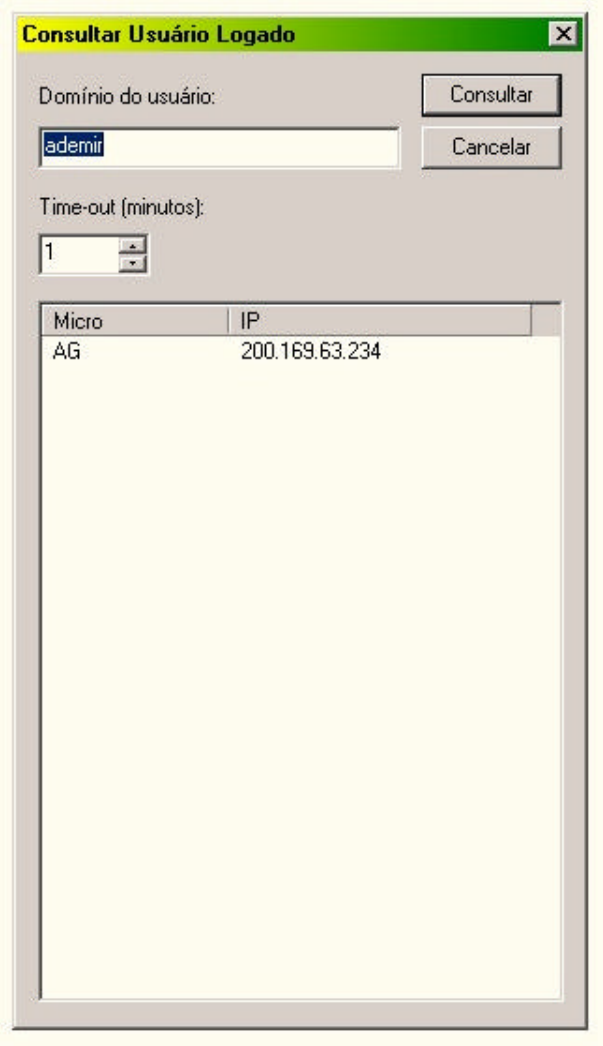

Figura 47 Consulta Usuário

- Função Z – Desativa o programa cliente. Se esta solicitação for efetivada, após validar uma senha de acesso a este recurso, será enviada uma mensagem a todos os clientes e os mesmos serão encerrados. Em princípio um cliente é ativado quando se liga o micro, fica sempre executando, até que a máquina seja desligada.

8.1.3 Programa gerador de Log de Conexão

No mecanismo de Comunicação em Grupo que foi usado para a aplicação descrita anteriormente, todas as informações relativas às transições de estado estão disponíveis

automaticamente. Assim para cada cliente que é ativado ou desativado, esta mudança nos membros do grupo se reflete em uma mensagem de transição de estado que é enviada pelo SPREAD para todos os membros do grupo.

Aproveitou-se esta facilidade e foi desenvolvido um programa para ficar executando continuamente no mesmo equipamento onde está o *daemon* do SPREAD. Este programa em ambiente LINUX, tem como objetivo gravar um log com registros onde serão informadas data e hora que cada cliente foi ativado e desativado. Com esta facilidade passamos a ter um log que nos mostra a hora que o micro foi ligado e a hora que o micro foi desligado. Como as máquinas que estão sendo avaliadas estão na mesma rede local, onde está o *daemon* do SPREAD não se tem problemas com particionamento de rede.

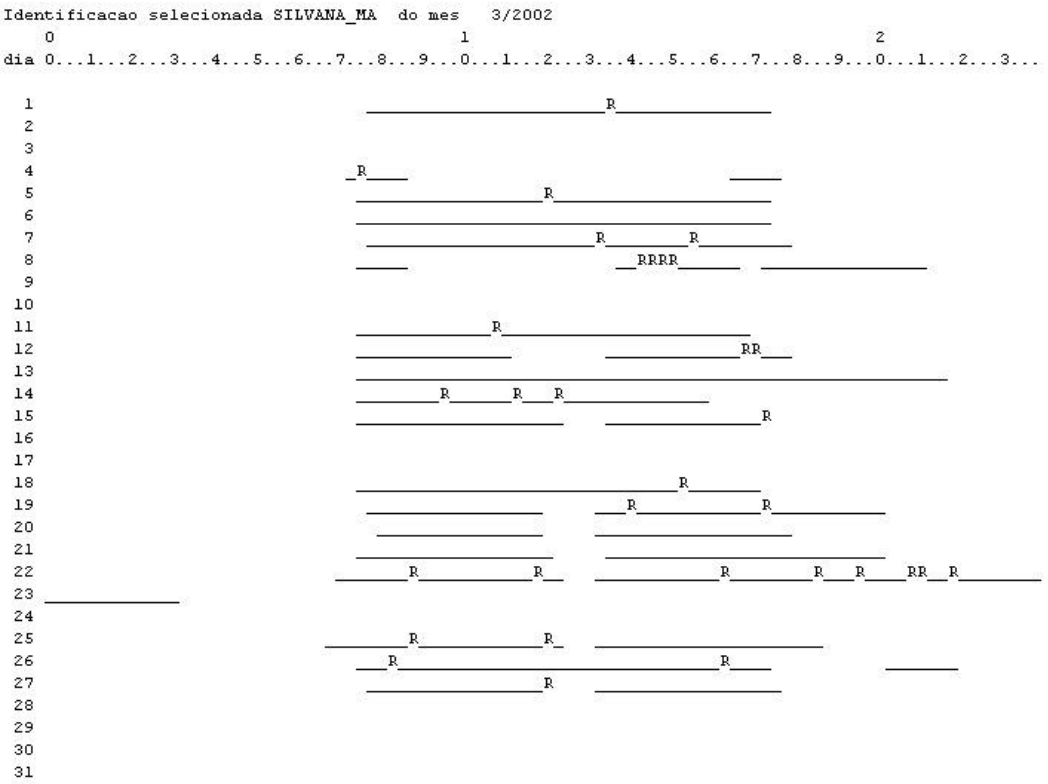

Figura 48 Relatório de Uso de um Micro

Desta forma um outro programa de visualização seleciona e tabula os registros de logs de conexão de um determinado micro, apresentando um gráfico de uso (máquina ligada) do mesmo em determinado período de tempo, conforme Figura 48. Em cada

linha é representado um dia do mês e a linha contínua mostra o tempo de uso no período de 0 até 24 horas sendo que cada segmento representa um quarto de hora. As letras “R” representam um indicativo de desliga/liga em horários muito próximos ou seja um *reboot* no sistema.. As informações gravadas em cada registro de log têm o seguinte formato:

- AAMMDD – Data no formato ano, mês e dia.
- HH:MM:SS – Hora no formato hora, minuto e segundo.
- E – Estado sendo C para conexão e D para desconexão.
- NOMEDOMICRO – Nome do micro que está sendo logado

Este programa gerador de log é ativado como um serviço na partida do sistema e somente será desativado quando o sistema for desligado. O arquivo de log é incremental e será rotacionado na rotina padrão de tratamento dos demais logs do sistema.

8.2 Experimentação Prática

Para a experimentação prática deste mecanismo de comunicação em grupo SPREAD, além do aplicativo de gerência de recursos em sistemas distribuídos descrito anteriormente, uma série de testes foram realizados em ambiente de rede real, nas instalações da UNIVALI, visando avaliar o comportamento do software quanto a escalabilidade e performance. Foram feitos testes com diversas configurações de rede, considerando um segmento único com um servidor único bem como com segmentos múltiplos de rede, interligados via WAN, e também múltiplos servidores rodando o software SPREAD. Esta arquitetura da rede usada será descrita no item seguinte, 8.2.1.

Todos os testes foram realizados usando sistema operacional LINUX e WINDOWS e os aplicativos desenvolvidos para este trabalho foram compilados e avaliados nos dois ambientes de sistema operacional. Os primeiros testes, durante o desenvolvimento dos aplicativos sempre envolveram um servidor LINUX executando o *daemon* SPREAD e algumas máquinas clientes (em torno de 5 a 10) existentes no mesmo ambiente físico do local de trabalho. Quando do desenvolvimento e testes em ambiente doméstico, sempre envolvia um notebook rodando LINUX e um micro de mesa rodando WINDOWS. Como compilador no WINDOWS foi usado o Visual C Versão 6.0 e no LINUX o GCC.

Como o objetivo do teste era avaliar a escalabilidade e a performance, foram desenvolvidos alguns programas que permitissem esta avaliação. A idéia principal era medir o tempo em um determinado cliente pertencente a um grupo, para o recebimento de N mensagens enviadas ao grupo com um determinado tamanho e usando diferentes tipos de entrega, quanto ao controle de ordenação, como confiável ou segura. Foram desenvolvidos os seguintes programas:

- FLOOENV – para o envio de mensagens a um grupo. Recebe como parâmetros o número de mensagens a enviar, o tamanho de cada mensagem e o tipo de ordenação da mensagem.
- FLOOREC – para o recebimento de mensagens, como cliente e membro do grupo. Uma vez executado, fica sempre ativo este programa, tratando cada lote de mensagens recebidas. No inicio de cada lote manda um registro com os dados deste teste corrente como data e hora, número de mensagens, tamanho da mensagem, tipo de ordenação e nome do cliente para um grupo chamado log. No final de cada lote de teste recebido, envia outro registro para o grupo log com os mesmos dados e a duração de tempo decorrido entre inicio do lote e final do lote recebido.
- LOGBM – para gravar os registros de log recebidos via grupo log. Assim cada programa cliente em cada lote medido, envia dois registros sendo um no início de recebimento do lote e outro no final de recebimento do lote.

Devido à falta de um relógio global, inicialmente foi usado o relógio do programa LOGBM que recebia todos os registros de log. Como se identifica que este tempo se mantém igual para todos os clientes, deduz-se que os registros de log são armazenados no servidor e entregues ao mesmo tempo para o programa que grava o log. Feita uma alteração no programa cliente, passa-se a trabalhar com o relógio de cada programa cliente. Assim no registro de início de recebimento de um lote, são enviadas a data e hora do cliente. No final de recebimento do lote é enviado um registro com a data e hora final de recebimento, o tempo decorrido entre o inicio e o fim, bem como o número real de mensagens recebidas pelo programa cliente.

As medidas de performance foram realizadas, variando o número de mensagens em cada lote e o tamanho das mensagens. As medidas avaliando escalabilidade foram feitas em um ambiente do 9 máquinas e comparadas com um segundo ambiente de 167 máquinas fazendo parte de um mesmo grupo.

Um exemplo da interligação, na Figura 49, mostra estes programas usados no teste de performance, FLOOENV, FLOOREC e LOGBM.

.

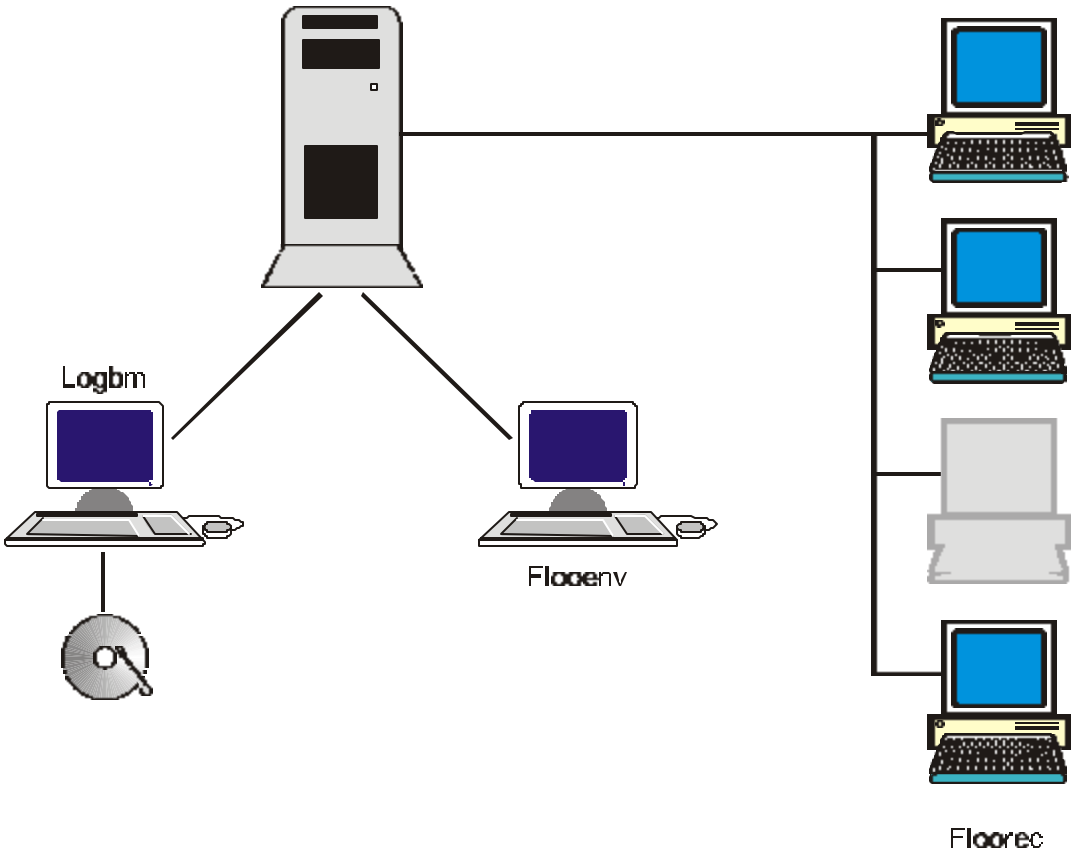

Figura 49 Interligação dos Programas FLOOENV, FLOOREC e LOGBM

8.2.1 Configuração da Rede

A rede normalmente usada para os testes iniciais é composta de 9 micros clientes em ambiente WINDOWS, e um servidor LINUX. As placas de rede são todas de 100 *Megabits* por segundo e todos os equipamentos estão ligados a um *switch* departamental caracterizando um ambiente de rede local ideal sem problemas de colisão, mesmo em grandes volumes de transmissão. Nos primeiros testes um programa cliente recebia a mensagem e mostrava o conteúdo recebido na tela. A demora em transmitir a informação recebida via rede para o dispositivo de vídeo era muito grande fazendo com

que a fila de mensagens no servidor, para este cliente, tenha alcançado o limite de área alocada. Nesta situação, o servidor como proteção desconecta o cliente. Este fato ocorria quando era enviado acima de mil mensagens. Posteriormente o programa cliente foi alterado para simplesmente receber as mensagens e contar quantas foram recebidas, mostrando na tela um totalizador de mensagens a cada 1000 mensagens recebidas. Desta forma a desconexão deixou de existir de forma tão freqüente, passando a ocorrer de forma muito mais esporádica. A Figura 50 mostra esta rede básica.

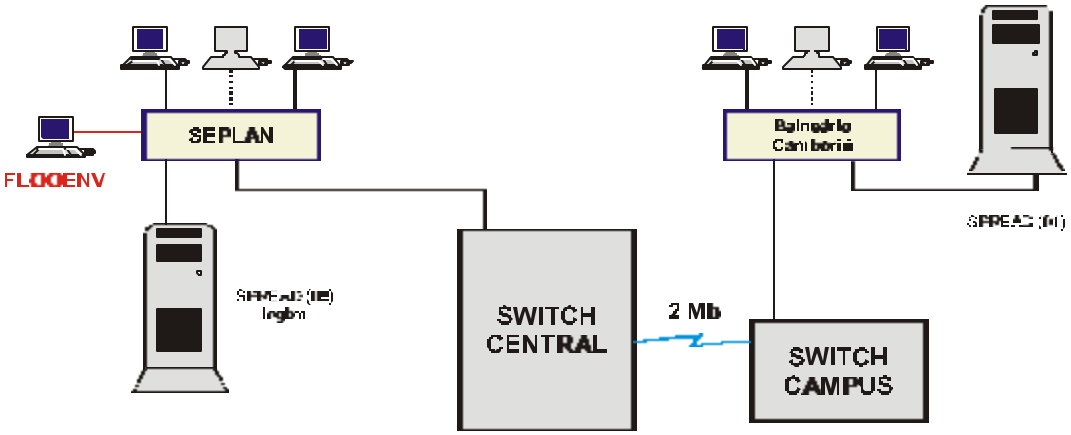

Figura 50 Rede Básica Usada no Teste Prático

Para o teste de volume, onde se atinge um total de 167 máquinas diferentes, diversos laboratórios em diferentes blocos são usados. Inclusive em diferentes Campi envolvendo ligação WAN com link de 2 Megabits. Usa-se um total de três servidores LINUX executando SPREAD com os clientes ligados a estes três servidores de forma distribuída. A Figura 51 mostra esta rede completa empregada no teste de volume.

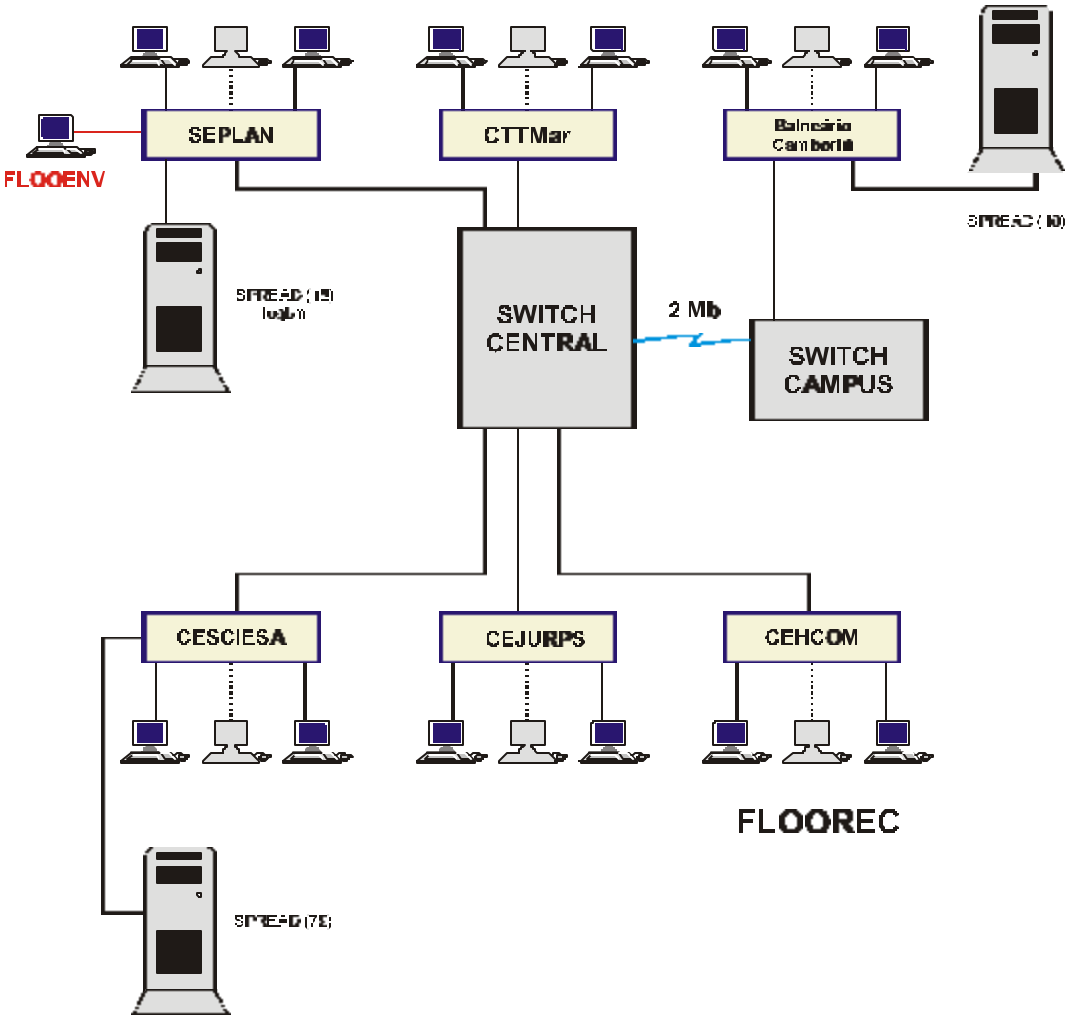

Figura 51 Rede Completa Usada no Teste Prático

Cabe ressaltar que durante o período total de realização do teste, alguns micros clientes tiveram o seu programa de recebimento de mensagens, FLOOREC, terminado devido a uma desconexão efetuada pelo SPREAD. Esta aplicação poderia tratar este código de erro informado pelo SPREAD tentando reconectar dentro de alguns instantes mas como o propósito era medir tempo de recebimento do lote completo, este tipo de erro não era recuperado e simplesmente terminava a aplicação cliente, sendo menos um resultado na amostragem final. Como a distância física não permite entre um teste e outro reativar estes programas que eram desconectados do SPREAD eles simplesmente ficaram desativados quando ocorreu este erro. É valido lembrar que este tipo de desconexão não é um erro e sim uma proteção para que uma determinada seção, ou seja, um cliente do SPREAD não venha a causar falta de memória na área para a fila de saída

deste cliente. Estas áreas de buffer para cada seção podem ser configuradas em tempo de compilação do software. Inicialmente, estavam configuradas para 1000 mensagens e posteriormente foram alteradas para 2000 mensagens (parâmetro MAX_SESSION em spread_params.h). Ao longo de todo o teste depois de recolher um total de 15.270 registros de informações no log sendo um de inicio do lote e um de final do lote para cada máquina que estava no grupo, um total de 37 máquinas tiveram o programa FLOOREC desativados por esta condição de erro. O Quadro 4 apresenta o número de máquinas por servidor SPREAD. Em cada linha tem-se a informação do número de máquinas por servidor no início e no final do teste efetuado. Cada cliente quando de sua ativação recebe como parâmetro um nome distinto para identificação e o endereço da porta e do *daemon* SPREAD ao qual se conecta.

Quadro 4 Número de Máquinas por Servidor SPREAD

SERVIDOR SPREAD	Número de máquinas	
	Início	Fim
Balneario Camboriu	40	40
CESCIESA	79	60
SEPLAN	48	30
	167	130

Esta mesma distribuição de todos os equipamentos usados no teste de volume, é apresentada no Quadro 5, agora distribuídos por local físico.

Quadro 5 Número de Máquinas por Local Físico

LOCAL FÍSICO	Número de máquinas	
	Início	Fim
Lab Letras	25	14
Lab CEJURPS	15	8
Lab CESCIESA	39	39
Lab CTTMar	40	21
Balneario Camboriu	40	40
SEPLAN	8	8
	167	130

8.2.2 Resultados das Medições

As medidas foram coletadas de forma automática, através da gravação de logs. As informações de data e hora, indicador de começo do lote “c” ou de termino do lote “t” identificação do micro e do servidor SPREAD onde o mesmo estava ligado, número de mensagens enviada no lote, tamanho da mensagem em bytes, tipo de ordenação da mensagem, comentário sobre o lote, hora de inicio no cliente ou hora de fim no cliente, com duração e número efetivo de mensagens recebidas eram gravadas em cada registro. Na Figura 52 temos um exemplo com alguns registros deste log.

```
200205 1 13:52:27 c @jul3#Seplan_lan_____________ c  10000    100 RELIABLE_MESS___ x11 13:51:25
200205 1 13:52:27 c @jul4#Seplan_lan_____________ c  10000    100 RELIABLE_MESS___ x11 13:51:26
200205 1 13:52:27 c @jul5#Seplan_lan_____________ c  10000    100 RELIABLE_MESS___ x11 13:50:33
200205 1 13:52:27 c @jul6#Seplan_lan_____________ c  10000    100 RELIABLE_MESS___ x11 13:52:36
200205 1 13:52:27 c @letras08#Seplan_____________ c  10000    100 RELIABLE_MESS___ x11 13:52:58
200205 1 13:52:27 c @letras10#Seplan_____________ c  10000    100 RELIABLE_MESS___ x11 13:53:23
200205 1 13:52:27 c @krabel@Seplan_n_____________ c  10000    100 RELIABLE_MESS___ x11 13:48:33
200205 1 13:52:27 c @dayse#Seplan__n_____________ c  10000    100 RELIABLE_MESS___ x11 14:01:29
200205 1 13:52:27 c @itabajara#Seplan____________ c  10000    100 RELIABLE_MESS___ x11 13:47:03
200205 1 13:52:27 c @silvano@Seplan_n____________ c  10000    100 RELIABLE_MESS___ x11 13:51:40
200205 1 13:52:27 c @jacques@Seplan_n____________ c  10000    100 RELIABLE_MESS___ x11 13:50:59
200205 1 13:52:27 c @letras08#Seplan_____________ c  10000    100 RELIABLE_MESS___ x11 13:53:51
200205 1 13:52:27 c @cynthia@Seplan______________ c  10000    100 RELIABLE_MESS___ x11 13:50:52
200205 1 13:52:27 c @victor@Seplan_______________ c  10000    100 RELIABLE_MESS___ x11 13:56:24
200205 1 13:52:27 c @giovani@Seplan______________ c  10000    100 RELIABLE_MESS___ x11 13:52:28
200205 1 13:52:27 c @jull@Seplan_an______________ c  10000    100 RELIABLE_MESS___ x11 13:48:14
200205 1 13:54:03 t @marcio@Neto_an______________ t  10000    100 RELIABLE_MESS___ x11 13:52:14    98  calc=  10000
200205 1 13:54:03 t @bc40@Neto_o_an______________ t  10000    100 RELIABLE_MESS___ x11 13:54:14    97  calc=  10000
200205 1 13:54:03 t @bc01@Neto_o_an______________ t  10000    100 RELIABLE_MESS___ x11 13:50:14    97  calc=  10000
200205 1 13:54:03 t @bc08@Neto_o_an______________ t  10000    100 RELIABLE_MESS___ x11 13:50:43    97  calc=  10000
200205 1 13:54:03 t @victor@Seplan_______________ t  10000    100 RELIABLE_MESS___ x11 13:50:01    97  calc=  10000
200205 1 13:54:03 t @jacques@Seplan______________ t  10000    100 RELIABLE_MESS___ x11 13:52:35    97  calc=  10000
200205 1 13:54:03 t @giovani@Seplan______________ t  10000    100 RELIABLE_MESS___ x11 13:54:05    97  calc=  10000
200205 1 13:54:03 t @dayse#Seplan_n______________ t  10000    100 RELIABLE_MESS___ x11 14:03:07    98  calc=  10000
200205 1 13:54:03 t @silvano@Seplan______________ t  10000    100 RELIABLE_MESS___ x11 13:53:17    97  calc=  10000
200205 1 13:54:03 t @ju01@Seplan_an______________ t  10000    100 RELIABLE_MESS___ x11 13:54:59    97  calc=  10000
200205 1 13:54:03 t @krabel@Seplan_______________ t  10000    100 RELIABLE_MESS___ x11 13:50:10    97  calc=  10000
200205 1 13:54:03 t @ju03#Seplan_n_______________ t  10000    100 RELIABLE_MESS___ x11 13:53:59    97  calc=  10000
200205 1 13:54:03 t @cynthia@Seplan______________ t  10000    100 RELIABLE_MESS___ x11 13:52:29    97  calc=  10000
200205 1 13:54:03 t @ju05#Seplan_an______________ t  10000    100 RELIABLE_MESS___ x11 13:58:20    97  calc=  10000
200205 1 13:54:03 t @ju06#Seplan_an______________ t  10000    100 RELIABLE_MESS___ x11 13:54:23    97  calc=  10000
200205 1 13:54:03 t @itabajara@Seplan____________ t  10000    100 RELIABLE_MESS___ x11 13:48:40    97  calc=  10000
200205 1 13:54:03 t @jul0#Seplan_plan____________ t  10000    100 RELIABLE_MESS___ x11 13:52:04    97  calc=  10000
200205 1 13:54:03 t @jul1#Seplan_plan____________ t  10000    100 RELIABLE_MESS___ x11 13:49:51    97  calc=  10000
200205 1 13:54:03 t @jul2#Seplan_plan____________ t  10000    100 RELIABLE_MESS___ x11 13:53:43    97  calc=  10000
200205 1 13:54:03 t @jul3#Seplan_plan____________ t  10000    100 RELIABLE_MESS___ x11 13:53:03    98  calc=  10000
200205 1 13:54:03 t @jul4#Seplan_plan____________ t  10000    100 RELIABLE_MESS___ x11 13:53:13    97  calc=  10000
200205 1 13:54:03 t @jul6#Seplan_plan____________ t  10000    100 RELIABLE_MESS___ x11 13:54:13    97  calc=  10000
200205 1 13:54:03 t @letras21#Seplan_____________ t  10000    100 RELIABLE_MESS___ x11 13:54:22    98  calc=  10000
```

Figura 52 Exemplo de Arquivo de Log

As medições ocorreram em diversos momentos, sendo que para cada um deles um parâmetro era variado. Para o parâmetro tipo de seqüência foram consideradas mensagens dos tipos confiáveis (*Reliable*) e mensagens do tipo segura (*Safe*). Dois tamanhos de mensagem foram escolhidos 100 bytes e 1000 bytes. O próximo parâmetro a variar foi o número de mensagens enviadas em cada bateria de testes, tendo sido

escolhidos os seguintes valores: 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000 e 10000. Assim o número total de execuções do programa que gerava as mensagens foi de 44 vezes (2 x 2 x 11) sendo que em cada execução para um total de 167 máquinas tínhamos 334 registros de log. Se nenhuma máquina tivesse sido desativada durante o teste, chegaríamos a um total de registros de log de 14.696 registros validos.

Outro fato importante a ser considerado é que o programa que envia as mensagens também pode eventualmente ser desconectado pelo SPREAD. Existe uma área de memória para recebimento de mensagens pelo SPREAD e que em nossa configuração estava definida para 1500 mensagens. Assim se por alguma razão, o servidor SPREAD não consegue dar vazão às mensagens entrantes, antes que tenha problema de memória, o SPREAD desconecta a aplicação que está causando esta sobrecarga. A variável que controla este limite é possível de ser alterada e tem o nome de WATER_MARK em spread_params.h. Como em algumas baterias de teste teve-se este problema, isto gerou um conjunto de registros de log apenas com o início sem os correspondentes registros de fim e estas rodadas foram ignoradas na amostragem. Em função disto teve-se em nosso arquivo um total final de 15.270 registros coletados.

As medições ocorreram em duas datas distintas, formando dois conjuntos de medidas sendo o primeiro para um ambiente simples de apenas 9 equipamentos e o segundo conjunto de medidas para um ambiente de 167 equipamentos. Os quadros que seguem, Quadro 6, Quadro 7, Quadro 8 e Quadro 9 mostram os valores obtidos para o ambiente com 9 micros clientes. Para todos os quadros tem-se uma identificação no cabeçalho com o tipo de mensagem, tamanho da mensagem em bytes e identificação do ambiente de teste. Nas linhas seguintes temos os dados de cada uma das baterias de testes realizadas, com as informações de :

- Número de mensagens enviadas
- Quantidade de amostras coletadas (quantas estações clientes responderam ao final de cada bateria de testes)
- Menor tempo total de transmissão do lote de mensagens
- Maior tempo total de transmissão do lote de mensagens
- Tempo médio calculado tomando como base todos os valores de tempo de resposta de todos os clientes.

Quadro 6 Medidas Rede Simples Mensagens 1000 Reliable

Mensagens do tipo				Reliable
Tamanho da mensagem				1000 bytes
Ambiente de rede com 9 micros				QR1S
número mensagens	quantidade amostras	tempo em segundos menor	maior	médio
100	9	1	2	1,66
200	9	3	4	3,33
300	9	5	5	5,00
400	9	6	7	6,44
500	9	8	8	8,00
1000	9	16	17	16,22
2000	9	31	32	31,77
3000	9	49	50	49,11
4000	9	66	67	66,44
5000	9	82	83	82,88
10000	9	159	159	159,00

Quadro 7 Medidas Rede Simples Mensagem 1000 Safe

Mensagens do tipo				Safe
Tamanho da mensagem				1000 bytes
Ambiente de rede com 9 micros				QR2S
número mensagens	quantidade amostras	tempo em segundos menor	maior	médio
100	9	1	2	1,77
200	9	3	4	3,55
300	9	5	5	5,00
400	9	6	7	6,11
500	9	8	8	8,00
1000	9	16	17	16,77
2000	9	33	33	33,00
3000	9	49	50	49,88
4000	9	64	65	64,88
5000	9	82	83	82,77
10000	9	160	161	160,44

Quadro 8 Medidas Rede Simples Mensagem 100 Reliable

Mensagens do tipo				Reliable
Tamanho da mensagem				100 bytes
Ambiente de rede com 9 micros				QR3S
número mensagens	quantidade amostras	tempo em segundos menor	maior	médio
100	9	0	1	0,55
200	9	1	2	1,22
300	9	1	2	1,88
400	9	2	3	2,44
500	9	3	4	3,22
1000	9	4	5	4,33
2000	9	13	14	13,22
3000	9	11	11	11,00
4000	9	25	26	25,66
5000	9	31	31	31,00
10000	9	61	62	61,66

Quadro 9 Medidas Rede Simples Mensagem 100 Safe

Mensagens do tipo				Safe
Tamanho da mensagem				100 bytes
Ambiente de rede com 9 micros				QR4S
número mensagens	quantidade amostras	tempo em segundos menor	maior	médio
100	9	0	1	0,66
200	9	1	12	1,33
300	9	2	2	2,00
400	9	2	3	2,33
500	9	3	4	3,44
1000	9	4	5	4,66
2000	9	13	14	13,44
3000	9	19	20	19,88
4000	9	25	26	25,88
5000	9	33	34	33,11
10000	9	64	65	64,55
100000	9	333	334	333,33

Na segunda etapa do teste as mesmas medidas foram também tomadas agora porém para um a ambiente de testes envolvendo 167 máquinas . Estes dados estão representados nos quadros Quadro 10, Quadro 11, Quadro 12 e Quadro 13.

Quadro 10 Medidas Rede Completa Mensagem 1000 Reliable

Mensagens do tipo				Reliable
Tamanho da mensagem				1000 bytes
Ambiente de rede com 167 micros				QR1F
número	quantidade	tempo em segundos		
mensagens	amostras	menor	maior	médio
100	167	1	3	1,88
200	167	3	7	3,94
300	167	4	9	5,78
400	167	6	11	7,46
500	167	8	14	9,38
1000	167	16	28	18,29
2000	167	33	54	36,10
3000	167	50	78	54,11
4000	167	66	104	72,22
5000	163	83	124	89,05
10000	152	166	174	166,64

Quadro 11 Medidas Rede Completa Mensagem 1000 Safe

Mensagens do tipo				Safe
Tamanho da mensagem				1000 bytes
Ambiente de rede com 167 micros				QR2F
número	quantidade	tempo em segundos		
mensagens	amostras	menor	maior	médio
100	163	2	4	2,47
200	163	3	6	4,09
300	163	5	8	5,73
400	163	6	11	7,43
500	163	7	13	9,10
1000	163	17	25	18,31
2000	163	33	49	35,26
3000	163	49	73	52,80
4000	163	65	97	70,52
5000	163	82	121	87,79
10000	154	166	186	168,33

Quadro 12 Medidas Rede Completa Mensagem 100 Reliable

Mensagens do tipo				Reliable
Tamanho da mensagem				100 bytes
Ambiente de rede com 167 micros				QR3F
número	quantidade	tempo em segundos		
mensagens	amostras	menor	maior	médio
100	152	0	2	0,35
200	152	1	2	1,19
300	152	1	36	8,12
400	152	2	3	2,42
500	152	2	4	2,83
1000	152	4	5	4,72
2000	152	7	9	7,95
3000	152	12	13	12,15
4000	152	15	17	15,44
5000	152	26	29	28,05
10000	73	97	98	97,39

Quadro 13 Medidas Rede Completa Mensagem 100 Safe

Mensagens do tipo				Safe
Tamanho da mensagem				100 bytes
Ambiente de rede com 167 micros				QR4F
número	quantidade	tempo em segundos		
mensagens	amostras	menor	maior	médio
100	73	0	1	0,63
200	73	1	2	1,19
300	73	1	2	1,35
400	152	2	4	2,43
500	152	2	4	2,67
1000	152	4	7	5,07
2000	152	7	9	8,14
3000	133	13	14	13,84
4000	133	19	21	19,93
5000	133	16	18	16,90
10000	133	33	35	33,95
100000	132	335	337	336,28

Todos os testes dos quadros anteriores foram feitos exatamente na seqüência apresentada. Se observarmos no início o total de micros clientes era de 167 e no final, apenas 132 pois diversos clientes foram desconectados do SPREAD devido a erros ocorridos. Se observarmos o final do Quadro 12 e o inicio do Quadro 13 vamos notar que tínhamos 152 micros, passou para 73 e voltou para 152 depois de algum tempo. O que ocorreu aqui foi exatamente um particionamento na rede onde todos os micros que estavam no servidor SPREAD do CESCIESA (Quadro 4) saíram do grupo e voltaram alguns minutos depois. Devido a alguma interrupção de comunicação entre a rede do CESCIESA e o restante da rede aqueles 79 micros gerenciados pelo SPREAD do CESCIESA ficaram isolados formando um outro grupo com os aplicativos esperando mensagens, nenhum programa FlOODER controlando estes clientes durante alguns minutos e quando a rede voltou a ter conexão física aqueles clientes voltaram a interagir e dando as respostas conforme pode ser visto no Quadro 13. Este fato ocorrido não foi identificado durante a aplicação dos testes pois todos os dados de medidas eram coletados automaticamente em disco para posterior análise. Quando da análise dos dados, identificado esta descontinuidade no número de máquinas, foi verificado que um conjunto completo de máquinas do servidor SPREAD do CESCIESA tinha perdido o contato em determinado horário. Conferindo também no log do servidor SPREAD que estava fisicamente na Seplan, pelo horário exato encontramos uma informação de transição de grupo, informando que em determinado instante, minutos após o particionamento estava sendo feito um *merge* entre os membros do grupo e novo total de membros passava a ser exatamente o mesmo conjunto de membros anterior ao particionamento. Assim pode-se conferir na prática uma das grandes vantagens do mecanismo de comunicação em grupo qual seja a propriedade de poder recompor o grupo se existir um particionamento da rede e posterior recomposição desta mesma rede.

8.2.3 Gráficos Comparativos

As medidas coletadas e apresentadas na seção anterior também deram origem a alguns gráficos comparativos, os quais serão detalhados a seguir.

Estas medidas referem-se ao tempo médio de transmissão das mensagens. Assim para cada lote de mensagens enviadas, foi calculado o tempo médio considerando a

soma de todos os tempos dividido pelo número exato de amostras em cada medição. Na Figura 53 tem-se uma comparação entre os tempos de transmissão na rede composta de 9 micros versus uma rede com 167 micros. Como pode ser visto a variação do tempo em segundos é mínima quando se aumenta a rede de 9 para 167 micros. Para este teste foram consideradas mensagens de 1000 *bytes* e ordenação do tipo *reliable*. Mesmo na amostragem mais demorada quando se envia 10.000 mensagens tem-se uma variação mínima.

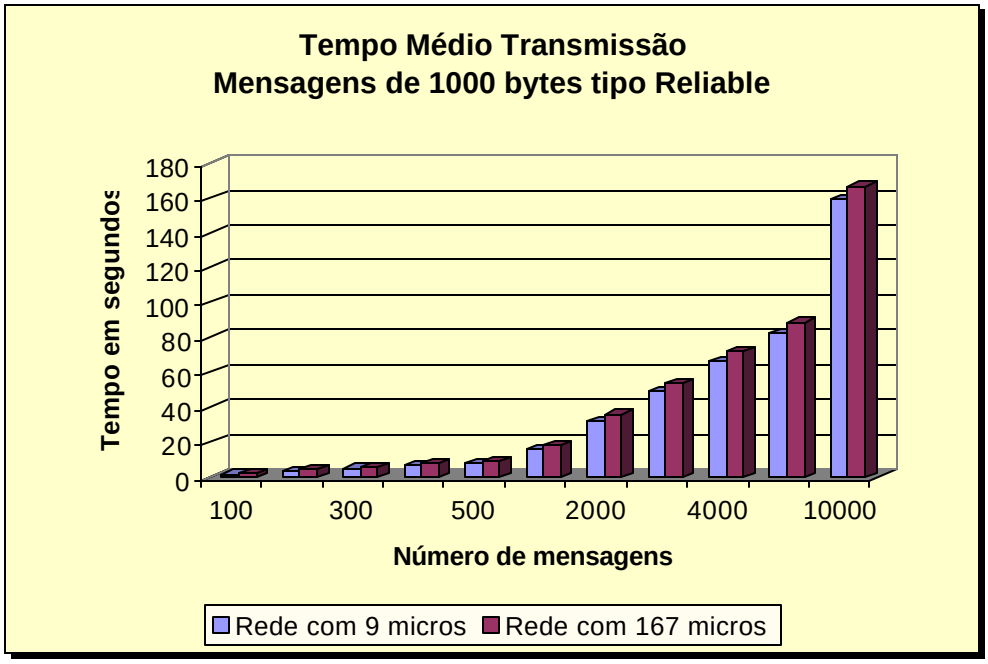

Figura 53 Tempo médio para mensagens em rede com 9 X 167 micros

Uma segunda comparação gráfica nos mostra os tempos quando se comparam diferentes formas de ordenação no envio das mensagens. O mecanismo de comunicação em grupo tem tratamento diferente com relação à ordem de entrega das mensagens ao aplicativo final. Como pode ser visto na

Figura 54 nesta medição específica não se tem variação sendo basicamente o mesmo tempo para mensagens com ordenação *reliable* ou *safe*. Uma explicação para este fato de não se ter variação é devido à característica do teste onde a fonte de envio era única e todas as mensagens eram enviadas seqüencialmente ordenadas pela origem que também era única. Para este teste foram usados como parâmetros mensagens de 1000 *bytes* sendo as medidas tomadas na rede de 167 micros.

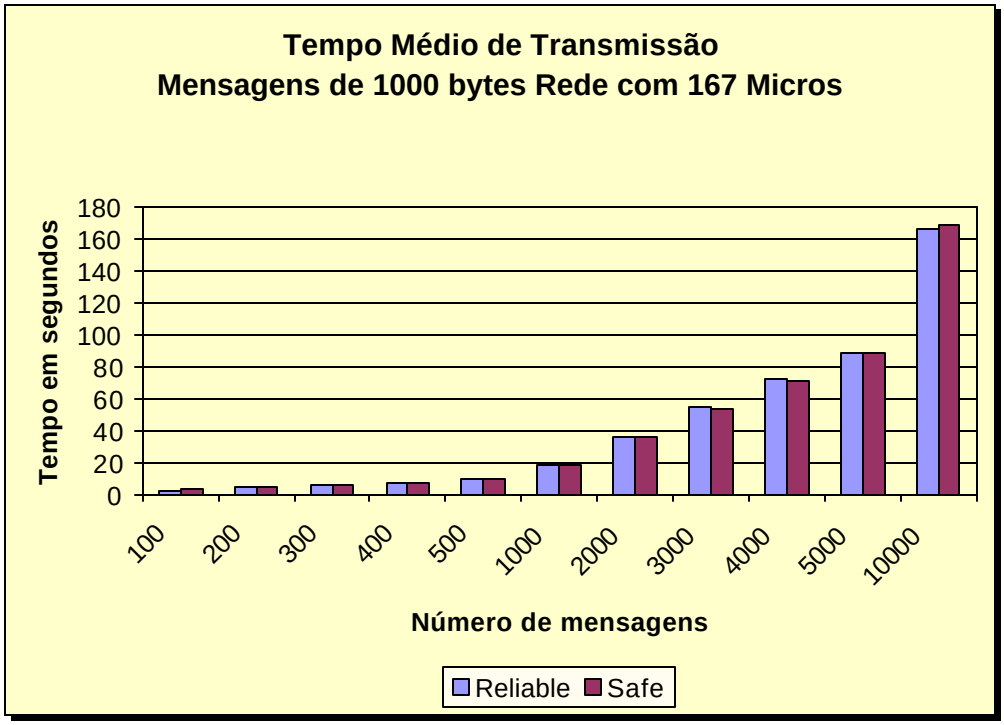

Figura 54 Tempo médio para mensagens *Reliable* X *Safe*

A terceira comparação gráfica, apresentada na Figura 55, nos mostra a variação de tempo quando se envia mensagens de 100 *bytes* comparativamente ao envio de mensagens de 1000 bytes. Embora o volume de informações enviado aumente de 10 vezes o tempo médio de transmissão das mensagens não aumenta na mesma proporção como pode ser visto no gráfico. Neste teste a ordenação das mensagens usada foi *reliable* no ambiente de rede com 167 micros.

Estes gráficos aqui mostrados no trabalho são provenientes dos quadros apresentados na seção anterior. As informações originais geradas e gravadas no arquivo de log, bem como as planilhas e gráficos originais, estão no CD em anexo a este trabalho.

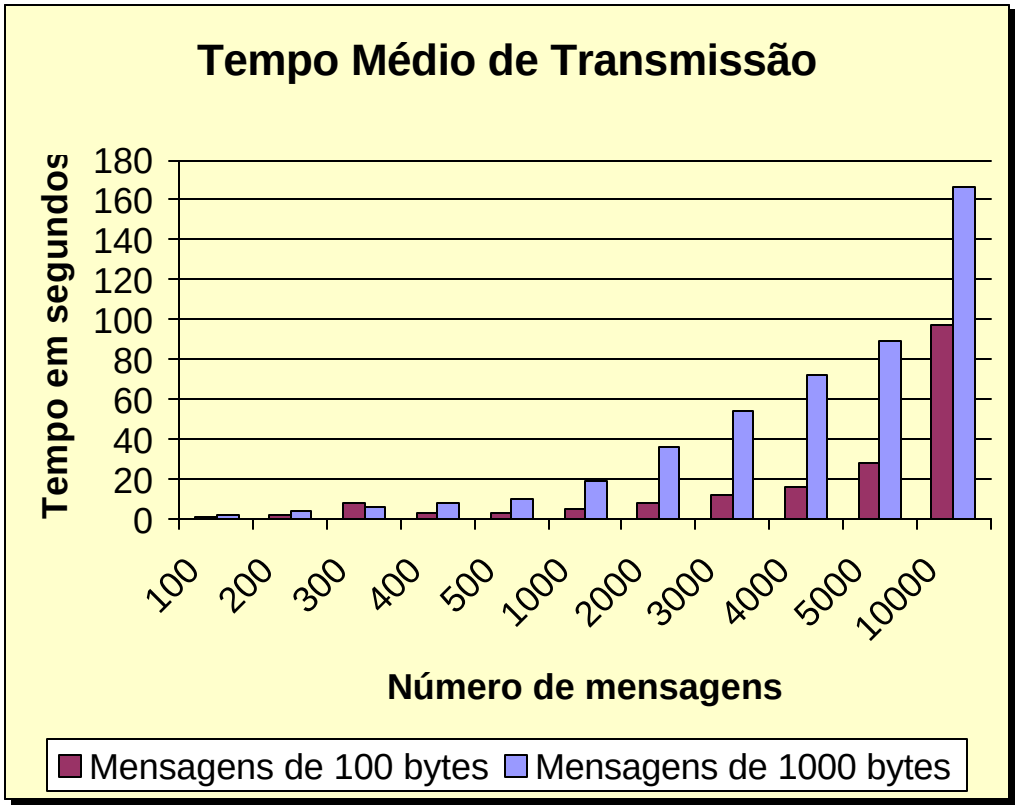

Figura 55 Tempo médio para mensagens de 100 X 1000 *bytes*

9 CONCLUSÃO

Após o levantamento preliminar de diversos mecanismos de Comunicação em Grupo que atualmente encontram-se disponíveis, a atenção ficou focada em três produtos: INTERGROUP, JGROUP e SPREAD. Um estudo mais detalhado foi efetuado nestes produtos pois a documentação dos mesmos era mais abrangente e permitia esta avaliação. Os dois primeiros estão voltados exclusivamente para o ambiente JAVA tendo sido implementados usando a linguagem JAVA. O SPREAD, selecionado para ser usado no teste de performance e na aplicação de gerenciamento de recursos em sistemas distribuídos, descritos no capítulo 8, está em ambiente multiplataforma, sendo desenvolvido em C, fortemente aderente a plataforma UNIX.

9.1 Quanto ao uso em Aplicações

Este estudo nos mostrou a importância dos mecanismos de comunicação em grupo pois com o uso deste recurso pode-se abstrair de quantas máquinas clientes tem-se no ambiente, quando do desenvolvimento de qualquer aplicação para ambientes de sistemas distribuídos. Fazer a mesma aplicação de gerenciamento de recursos aqui desenvolvida como exemplo de uso de comunicação em grupo, sem esta ferramenta, implicaria em uma preocupação de dimensão proporcional ao tamanho da rede que se quer gerenciar tornando o aplicativo praticamente impossível quando a escala atingir algumas dezenas de máquinas. Todos os tratamentos de erros em ambiente WAN ficariam enviáveis para uma aplicação *stand-alone*.

Não apenas para aplicações relativas ao sistema operacional mas também para aplicações comerciais é possível e plenamente viável o uso dos mecanismos de comunicação em grupo. Como exemplo podemos citar aplicações de replicação de arquivos de log, replicação de servidores WEB e replicação de banco de dados.

9.2 Quanto a Performance e Escalabilidade

Em relação à performance e escalabilidade, pode-se observar que o ambiente SPREAD tem ótima escalabilidade, produzindo tempos de resposta plenamente adequados durante os testes com a variação do número de componentes envolvidos.

Tanto no volume de pequena rede com 9 micros ou no teste chamado de rede completa com 167 micros tive-se um tempo de resposta muito semelhante o que nos permite concluir que o SPREAD funciona de forma eficiente quando se aumenta de dezenas para centenas de máquinas. Naturalmente deve-se considerar que no teste de pequena rede foi usado um *daemon* SPREAD e no teste de rede completa usamos três *daemon* SPREAD. Este mecanismo de comunicação em grupo escala muito bem, bastando para isso apenas ir acrescentando novos *daemon* SPREAD em nossa rede à medida que o número de clientes for incrementado em algumas dezenas.

Outra consideração a ser feita sobre performance está relacionada com o tipo de ordenação das mensagens. Embora o SPREAD trabalhe com diferentes tipos de ordenação de mensagens tais como sem ordem, FIFO, CAUSAL e SAFE para o tipo de teste realizado, basicamente não se teve diferença de tempo. O processo de ordenação faz sentido quando tem se diversas fontes de envio simultâneo para o grupo. Em nosso teste como era única fonte de envio para o grupo deixa de existir a importância desta característica de ordenação.

9.3 Futuros Trabalhos e Melhorias

Uma sugestão de melhoria no produto, seria a possibilidade de reconfiguração dinâmica no SPREAD. Hoje quando se inclui um novo servidor na rede SPREAD todos os nós têm que ser desativados e reativados. A idéia seria usar os algoritmos de particionamento e reagrupamento de membros do grupo já existentes, para esta passagem dinâmica dos clientes de um servidor que está sendo desativado para um novo servidor já reconfigurado. Assim uma nova configuração seria iniciada em qualquer ponto da rede e através de um comando de gerência, receberia toda a partição do servidor que precisa de manutenção.

Quanto a futuros desenvolvimentos, pode-se pensar em diversas aplicações usando este recurso de comunicação em grupo, tais como, replicação de servidores de banco de dados, servidores WEB, programas que simulam um quadro branco, replicando imagens em uma ambiente de ensino a distância e qualquer outra aplicação que envolva a comunicação de um aplicativo para N clientes.

Complementando a aplicação de gerenciamento de recursos em sistemas distribuídos apresentada neste trabalho fica a sugestão de refazer a mesma usando a

linguagem de programação e o ambiente JAVA, considerando que o SPREAD tem as suas funções implementadas em classes JAVA, permitindo assim que o programa de consulta possa ser executado em ambiente WEB.

10 REFERÊNCIAS BIBLIOGRÁFICAS [1]

AMIR, Y. **Replication using Group Communication over a Partitioned Network** Ph.D. Thesis. Jerusalem: Institute of Computer Science, The Hebrew University of Jerusalem, 1995.

AMIR, Y.; STANTON, J. **The SPREAD Wide Area Group Communication System.** Baltimore: Technical report, Center of Networking and Distributed Systems, Johns HOPkins University,1998. Disponível em <http://www.cnds.jhu.edu/publications/>. Acesso em: agosto de 2001.

ATTIYA H.; WELCH J. **Distributed Computing:** Fundamentals, Simulations and Advanced Topics. London: McGraw-Hill, 1998.

BERKET, K., **The InterGroup Protocols:** Scalable Group Communication for the Internet. Dissertation, University of California – USA 2000. Disponível em <http://www-itg.lbl.gov/InterGroup>. Acesso em: dezembro de 2001.

BIRMAN, K; RENESSE, R. Van. **Reliable Distributed Computing with the ISIS Toolkit**, Los Alamitos: IEEE Computer Society Press, 1994.

BIRMAN, K. **Projeto ISIS Pagina do professor BIRMAN**, disponível em <http://www.cs.cornell.edu/Info/Department/Annual95/Faculty/Birman.html> e <http://www.cs.cornell.edu/Info/Projects/ISIS/>. Acesso em: março de 2002.

BROADCAST **Technical Report Series:** Basic Research On Advanced Distributed Computing from Algorithms to SysTems, 1997. [online] Disponível em <http://www.newcastle.research.ec.org/broadcast>. Acesso em: maio de 2002.

BUYYA, R. **High Performance Cluster Computing:** Architecture and Systems, vol 1. pags 3-45, New Jersey: Prentice Hall, 1999.

[1] Seguiu-se a NBR 6023/2000 que substitui a NBR 6023/1989. Nas citações seguiu-se NBR 10520/2001 que substitui a NBR 10520/1992.

COULOURIS G.; DOLLIMORE J.; KINDBERG T. **Distributed Systems Concepts and Design.** 3.ed. Harlow, England: Addison Wesley, 2001.

DIETZ, Hank **LINUX Parallel Processing HOWTO**, 1998 [online]. Disponível em <http://yara.ecn.purdue.edu/~ppLINUX/PPHOWTO/pphowto.html>. Acesso em: outubro de 2001.

DOLEV, D.; MALKI, D. **The Transis Approach to High Availability Cluster Communication. New York:** Communications of the ACM, 39(4), April 1996.

EMMERICH W. **EngineeRING Distributed Objects**. Chichester: John Wiley & Sons Ltd, 1999.

EZHILCHELVAN, P.; MACEDO, R.; SHRIVASTAVA, A. **NEWTOP**: A Fault-tolerante Group Communication Protocol. Vancouver: Proceedings of the 15th IEEE International Conference on Distributed Computing Systems, pag 296-306, maio 1995.

GALLI, D.L. **Distributed Operating Systems Concepts and Practice.** New Jersey: Prentice Hall, 1998.

HAYDEN, M. **The Ensenble System.** PhD Thesis, Ithaca,NY: Department of Computer Science, Cornell University, January 1998.

KAASHOEK, M.F.; TANENBAUM, A.S. **Group Communication in AMOEBA and its applications**, Distributed Systems EngineeRING Journal, vol 1, pp 48-58, julho1993 [online]. Disponível em <http://www.cs.vu.nl/vakgroepen/cs/amoeba_papers.html>. Acesso em: janeiro de 2002.

KOCH, R.R. **The Atomic Group Protocols:** Reliable Ordered message delivery for ATM networks. PhD Dissertation, Department of Electrical and Computer EngineeRING, Santa Barbara: University of California, December 2000.

MALLOTH, C. **Conception and Implementation of a Toolkit for Building Fault-Tolerant Distributed Applications in Large-Scale Networks.** PhD thesis, Lausanne, Switzerland: Ecole Polytechnique Federale de Lausanne, September 1996.

MONTRESOR, A. **System Support for Programming Object-Oriented Dependable Applications in Partitionable Systems.** Technical Report UBLCS-2000-10 Department of Computer Science, Bologna: University of Bologna, 2000. Disponível em <http://www.cs.unibo.it>. Acesso em: dezembro de 2001.

MOSER, L. E.; AMIR, Y.; MELLIAR-SMITH, P.M.; AGARWAL, D. A. **Extended Virtual Synchrony.** Poznan: Proceedings of the 14th IEEE International Conference on Distributed Computing Systems, pags 56-65, junho 1994. Los Alamitos: IEEE Computer Society Press, 1994.

MOSER, L. E.; MELLIAR-SMITH, P.M.; AGARWAL, D.A.;BUDHIA, R.;LINGLEY-PAPADOULOS, C. **Totem:** A Fault-Tolerant Group Communication System. New York: Communications of the ACM, 39(4), April 1996.

MULLENDER S. **Distributed Systems.** 2.ed. New York: Addison-Wesley, 1993.

MYRICOM **Myrinet Overview**, 2001 Disponível em <http://www.myri.com/myrinet/overview/index.html>. Acesso em: 01 fev 2002.

OMG – Object Management Group. **The Common Object Request Broker: Architecture and Specification**, *Rev 2.3* OMG Inc., Framingham, Mass., March 1998.

OPEN Group **DCE Overview**, 1999. [online] Disponível em <http://www.opengroup.org/dce/info/papers/tog-dce-pd-1296.htm>. Acesso em: janeiro de 2002.

RADAJEWSKI, J., EADLINE D. **Beowulf HOWTO.** 1998. [online] Disponível em <http://www.LINUXvoodoo.com/howto/HOWTO/Beowulf-HOWTO/Beowulf-HOWTO-4.html>. Acesso em: outubro de 2001.

RENESSE, R. van; BIRMAN, K.P.; MAFFEIS, S. **Horus:** A Flexible Group Communication System. New York: Communications of ACM, 39(4): 76-83, April 1996.

SILBERSCHATZ, A., GALVIN, P.B. **Sistemas Operacionais Conceitos.** 5.ed., São Paulo: Prentice-Hall, 2000.

STALLINGS, W. **Operating Systems Internals and Design Principles.** 3.ed. New Jersey: Prentice-Hall, 1998.

SUN Microsystems. **Java Remote Method Invocation Specification**, *Rev 1.50*. 1998, Sun Microsystems, Inc., Mountain View, California, October 1998.

TANENBAUM A. S. **Sistemas Operacionais Modernos,** Rio de Janeiro: Prentice-Hall do Brasil Ltda, 1995.

VERÍSSIMO, P.; RODRIGUES, L. **Distributed Systems for System Architects.** Norwell, Massachusetts: Kluwer Academic Publishers, 2001.

WHETTEN, B.; MONTGOMER, Y., T.; KAPLAN, S. **A High Performance Totally Ordered Multicast Protocol**. Proceedings of the International WorksHOP on Theory and Practice in Distributed Systems, pag 33-57, Dagstuhl Castle, Alemanha, Setembro 1994.

ANEXO A Descrição do Conteúdo do CD que Acompanha este Trabalho

RELAÇÃO DE DIRETÓRIOS NO CD

_Textos	Arquivos de textos que foram criados tais como a dissertação (__dm01.doc) e artigos alem de pdfs correspondentes
Alberto_montresor	Documentação completa do JGROUP, com arquivos fontes em diversas versões, cópia das paginas da Universidade de Bolonha
Atomic	Tese de Ruppert R. Koch, The Atomic Group Protocols: Reliable Ordered Message Delivery for ATM network
Broadcast	Dezenas de artigos sobre BROADCAST, da pagina da universidade NewCastle - Contem abstracts e artigos completos com link para estes pesquisadores: Ecole Polytechnique Fédérale de Lausanne (CH) - André Schiper IMAG-LSR Grenoble (F) - Sacha Krakowiak INESC Lisboa (PT) - Paulo Veríssimo INRIA Rocquencourt (F) - Marc Shapiro IRISA Rennes (F) - Michel

	Banâtre, Michel Raynal Università di Bologna (I) - Ozalp Babaoglu University of Newcastle (UK) - Santosh Shrivastava Universiteit Twente (NL) - Sape Mullender
Cgeyer	Material do Cláudio Geyer fornecido no CBCOMP 2001. Apresentação dos cursos: Programação Distribuída e Paralela; Conceitos básicos de programação com objetos distribuídos; JAVA Threads e concorrencia; JAVA RMI; JAVA Sockets; Ordem de eventos em sistemas distribuídos; Algoritmos de PROBE/ECHO Difusão em uma rede; Algoritmos de eleição; Algoritmos Paralelos. DECK e artigos sobre DECK (UFRGS)
Diversos	Aqui um conjunto genérico de arquivos que não foram classificados em outros diretórios... Temos entre outros apressentação PPP sobre P2P, Capitulo UM do livro sobre Cluster do BUYYA, GFS Storage Cluster da SISTINA, Apresentação DIPC-MOSIX, Livro de C e C++ Orientação a Objetos de Sergio Barbosa Vilas-Boas UFRJ, alguns artigos, LATEX e documentação, Slides SGL-ISCC01, Slides palestra do Jack Stankovic e Ken Birman
Ensemble	THE ENSEMBLE SYSTEM, Dissertação de Mark Garland Hayden, Cornell University, 1998
Figuras	Figuras que foram usadas neste trabalho, Avaliação de Mecanismos de Comunicação em Grupo para Ambientes WAN
InterGroup	Todo o material sobre InterGroup do Karlo Berket, Lawrence Berkeley National Laboratory
Logs	Arquivos logs obtidos durante experimentação e planilhas geradas a partir dos arquivos log
Mosix	Artigo Linux clustering with MOSIX presented by developerWorks ibm/com/developerworks
Newtop	Artigos, NEWTOP: A Total Order Multicast Protocol Using Causal Blocks de Macedo, Ezhilchelvan e Shrivastava e NEWTOP: A Fault-Tolerant Group Communication Protocol de Ezhilchelvan, Macedo e Shrivastava; University of Newcastle
Ngc_2001	Índice dos artigos apresentados no terceiro NGC Network Group Communication em 2001 ocorrido em Londres (Nov/2001). O I NGC foi em Pisa - Italia 1999, o II NGC foi em Stanford, CA,USA 2000
Phoenix	Tese de Christoph Peter Malloth, Escola Politecnica Federal de Zurich, Conception and implementation of a toolkit for building fault-tolerant distributed applications in large scale networks
Ppt	Apresentação em Power Point usada na defesa da dissertação Avaliação de Mecanismos de Comunicação em Grupo para Ambientes WAN em 18/06/2002
Programas	Todos os programas fontes desenvolvidos neste trabalho, Avaliação de Mecanismos de Comunicação em Grupo para Ambientes WAN
Programas_executaveis	Todos os programas executaveis deste trabalho, Avaliação de

	Mecanismos de Comunicação em Grupo para Ambientes WAN, em ambiente WINDOWS e um sub diretorio com os executaveis para LINUX
RMP	RMP - Reliable Multicast Protocol, um paper sobre o mesmo, A HIGH PERFORMANCE TOTALLY ORDERED MULTICAST PROTOCOL, DE Brian Whetten (University of California at Berkeley), Todd Montgomery (West Virginia University) e Simon Kaplan (University of Illinois at Champaign-Urbana).
Robert_van_renesse	Um paper de Robbert van Renesse, Cornell University, Scalable and Secure Resource Location
SPREAD_diversos	Diversos documentos em PDF, artigos e users guide
SPREAD_mail_list	Uma relação de todas as mensagens da lista SPREAD desde 06/2000 até 06/2002
SPREAD_versões_downloac	Diversos arquivos tar, diferentes versões, com distribuições em binario e fontes do SPREAD, alem de spreadlogd e secure SPREAD
Spread-bin-3.16.2	SPREAD binários de todas as plataformas, ultima versão 2.16.2
Spread-src-3.16.2	SPREAD fontes da última versão 3.16.2 com makefile para diferentes plataformas
TRANSIS	Documentação sobre o TRANSIS, A Framework for Partitionable Membership Service, por Danny Dolev, Dalia Malki e Ray Strong, The Hebrew University of Jerusalem; The Transis Approach to High Availability Cluster Communication, por Danny Dolev e Dalia Malki; An Asynchronous Membership Protocol that Tolerates Partitions, por Danny Dolev, Dalia Malki e Ray Strong

HP - Páginas do Prof. Ademir Goulart, usadas no Curso de Ciência da Computação - UNIVALI

Programação C	Home Page da disciplina de linguagem de Programação C
Sistemas Distribuídos	Home Page da disciplina de Sistemas Distribuídos

Em caso de dúvida entre em contato pelo email:

ademir@univali.br